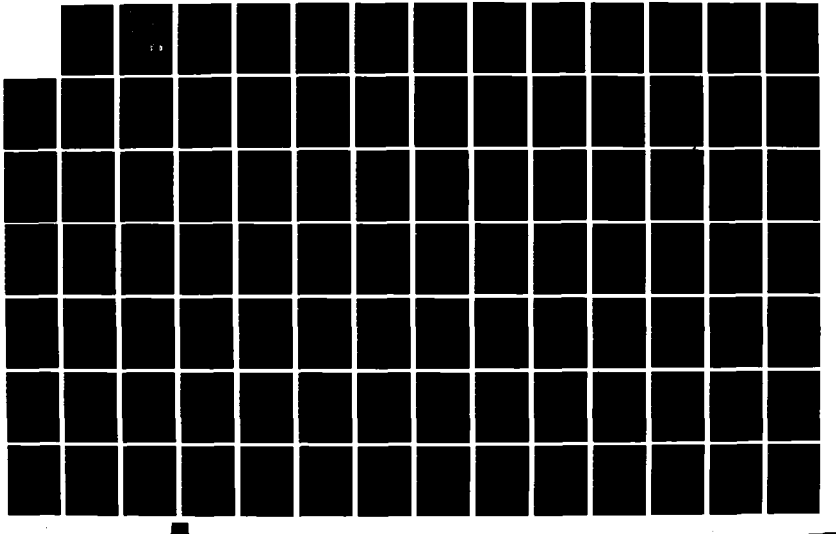
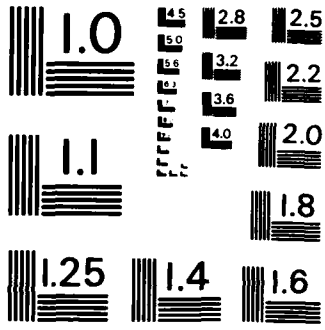
AD-A161 351 SPECIAL PURPOSE COMPUTER ARCHITECTURE FOR LU FACTORIZATION OF PARTITIONED SYSTEMS(U) ILLINOIS UNIV AT URBANA COORDINATED SCIENCE LAB K I LUI AUG 85 1/2

UNCLASSIFIED R-1015 N00014-84-C-0149 F/G 9/2 NL



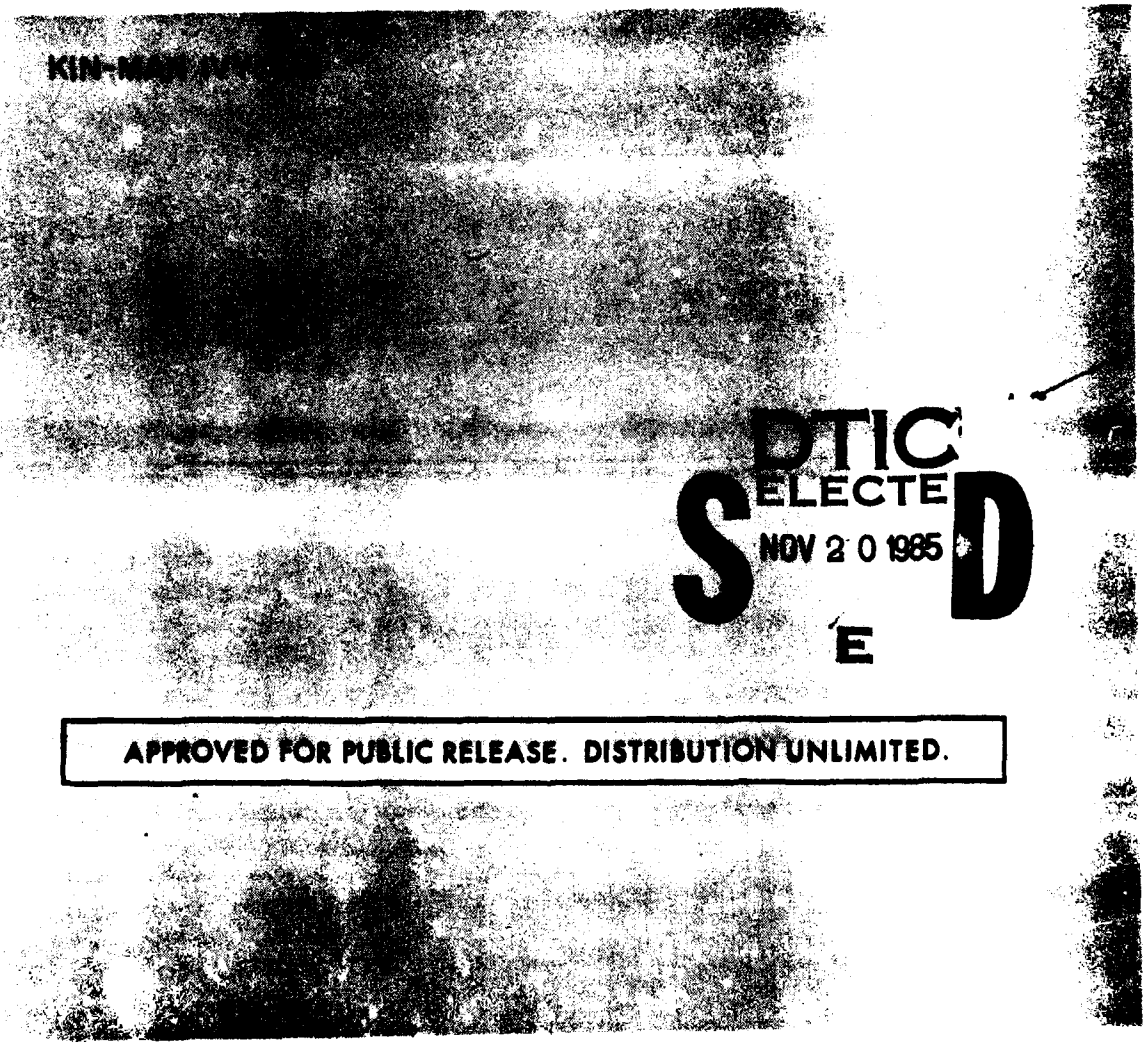


REPORT R-1015 AUGUST, 1984 UILU-ENG 84-2209

12

COORDINATED SCIENCE LABORATORY

SPECIAL PURPOSE COMPUTER ARCHITECTURE FOR LU FACTORIZATION OF PARTITIONED SYSTEMS



UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

AD-A166 357

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION: UNCLASSIFIED	1b. RESTRICTIVE MARKINGS: None
2a. SECURITY CLASSIFICATION AUTHORITY: N/A	3. DISTRIBUTION/AVAILABILITY OF REPORT: Approved for public release, distribution unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE: N/A	
4. PERFORMING ORGANIZATION REPORT NUMBER(S): R-1015 UILU-ENG 84-2209	5. MONITORING ORGANIZATION REPORT NUMBER(S): N/A
6a. NAME OF PERFORMING ORGANIZATION: Coordinated Science Laboratory, University of Illinois; 6b. OFFICE SYMBOL (If applicable): N/A	7a. NAME OF MONITORING ORGANIZATION: Office of Naval Research
6c. ADDRESS (City, State and ZIP Code): University of Illinois at Urbana-Champaign, 1101 West Springfield Ave., Urbana, IL 61801	7b. ADDRESS (City, State and ZIP Code): 800 N. Quincy Street, Arlington, VA 22217
8a. NAME OF FUNDING/SPONSORING ORGANIZATION: Joint Services Electronics Program; 8b. OFFICE SYMBOL (If applicable): N/A	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER:
8c. ADDRESS (City, State and ZIP Code): 800 N. Quincy Street, Arlington, VA 22217	10. SOURCE OF FUNDING NOS.

PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.	WORK UNIT NO.
N/A	N/A	N/A	N/A

11. TITLE (Include Security Classification): SPECIAL PURPOSE COMPUTER ARCHITECTURE FOR LU FACTORIZATION OF PARTITIONED SYSTEMS

12. PERSONAL AUTHOR(S): LUI, KIN-MAN IVY

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Yr., Mo., Day)	15. PAGE COUNT
Interim Technical, final	FROM Aug. '83 TO Aug. '84	August 1984	96

16. SUPPLEMENTARY NOTATION: N/A

17. COSATI CODES		
FIELD	GROUP	SUB. GR.

18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number): Integrated circuits, simulation; LU factorizaion; partitioned Systems; systolic arrays; wavefront array processors; Parallel Processing.

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

The simulation of large-scale integrated circuits requires a considerable amount of computation time using the currently available circuit simulation programs like SPICE. One of the bottlenecks of these simulation programs is in solving these systems of linear equations using LU factorization. This thesis explores the idea of a nested clustering algorithm to partition the matrices into bordered block diagonal form in order to partition the matrices into bordered block diagonal form in order to facilitate parallel processing. In addition, an architecture combining both the systolic array and the wavefront array processors is proposed to perform the LU factorization of the partitioned system using highly concurrent parallel processor arrays.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION
UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ OTIC USERS ☐	UNCLASSIFIED

22a. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE NUMBER (Include Area Code)	22c. OFFICE SYMBOL

SPECIAL PURPOSE COMPUTER ARCHITECTURE FOR LU FACTORIZATION OF PARTITIONED SYSTEMS

BY

KIN-MAN IVY LUI

B.S., University of California, Berkeley, 1983

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Electrical Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 1984

Urbana, Illinois

ABSTRACT

The simulation of large-scale integrated circuits requires a considerable amount of computation time using the currently available circuit simulation programs like SPICE. One of the bottlenecks of these simulation programs is in solving these systems of linear equations using LU factorization. This thesis explores the idea of a nested clustering algorithm to partition the matrices into bordered block diagonal form in order to facilitate parallel processing. In addition, an architecture combining both the systolic array and the wavefront array processors is proposed to perform the LU factorization of the partitioned system using highly concurrent parallel processor arrays.

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my advisor, Professor I.N. Hajj for his valuable guidance and expert assistance during the course of this research and the preparation of this thesis. In addition, I wish to thank Professor T.N. Trick, my co-advisor, for his helpful discussions and suggestions in this research. I would like to thank Miranda Hung for her helpful discussions. Also, thanks are given to Susan Lefferts, Eric Peterson and Beth Piver for helping me in the drawing of the figures.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

With the advances in VLSI technology, there is tremendous demand for efficient circuit simulators and other computer-aided design tools. One of the factors limiting the design time of VLSI circuits is the slowness of circuit simulation programs. Conventional circuit simulators , for example, SPICE [18], were designed initially for the cost-effective analysis of circuits containing a few hundred transistors or less. Because of the need to verify the performance of larger circuits, many users have used programs like SPICE [18] and have successfully simulated circuits containing thousands of transistors despite the cost. Because these circuit simulation programs are slow, designers are forced to use less accurate models and to make assumptions during the simulations in order to save computation cost. At the circuit level, the electrical behavior of the design is modelled in terms of algebraic-differential equations. The use of implicit integration methods, together with modified Newton iterative methods for solving the algebraic-differential equations representing the circuit model, has been found to give reliable numerical solutions, and has thus been implemented in many circuit simulators, such as SPICE [18].

A number of approaches have been used to improve the performance of conventional circuit simulators for the analysis of large circuits. The time required to evaluate complex device model equations

has been reduced by using table-lookup models [19]. Techniques based on special purpose microcodes have been investigated for reducing the time required to solve sparse linear systems arising from the linearization of the circuit equation [20]. Node tearing techniques [5,15] have also been used to exploit circuit regularity by bypassing the solution of subcircuits whose state is not changing and to exploit the vector-processing capabilities of high performance computers such as the Cray-1 used in the simulation program CLASSIE [21]. The advent of VLSI technology has made the cost-effective design of special purpose machines possible. Examples of these machines are the Yorktown Simulation Engine (YSE) for logic simulation [22], systolic arrays [4] and the wavefront array processors [2]. Special purpose machines have also been proposed for the solution of linear systems of equations [2]. Most of these machines limit the size of the operand matrix except for the one designed by Pottle [7]. Special matrix structures such as the Bordered Block Diagonal Form (BBDF) are discussed in Blossom [1].

The procedure involved in a standard circuit simulation is shown in Fig. 1 [16]. The majority of the time spent to run a circuit simulation can be lumped into two categories: the time required to solve the system of sparse linear equations, SOLVE (steps(7) and (8)) and the time required to form the entries of A and b in Ax=b, FORM (steps(5) and (6)). These two steps are repeated over and over again. As seen in Fig. 2, for small circuits ($N < 20$), the majority of the solution time is spent performing FORM. However, when the

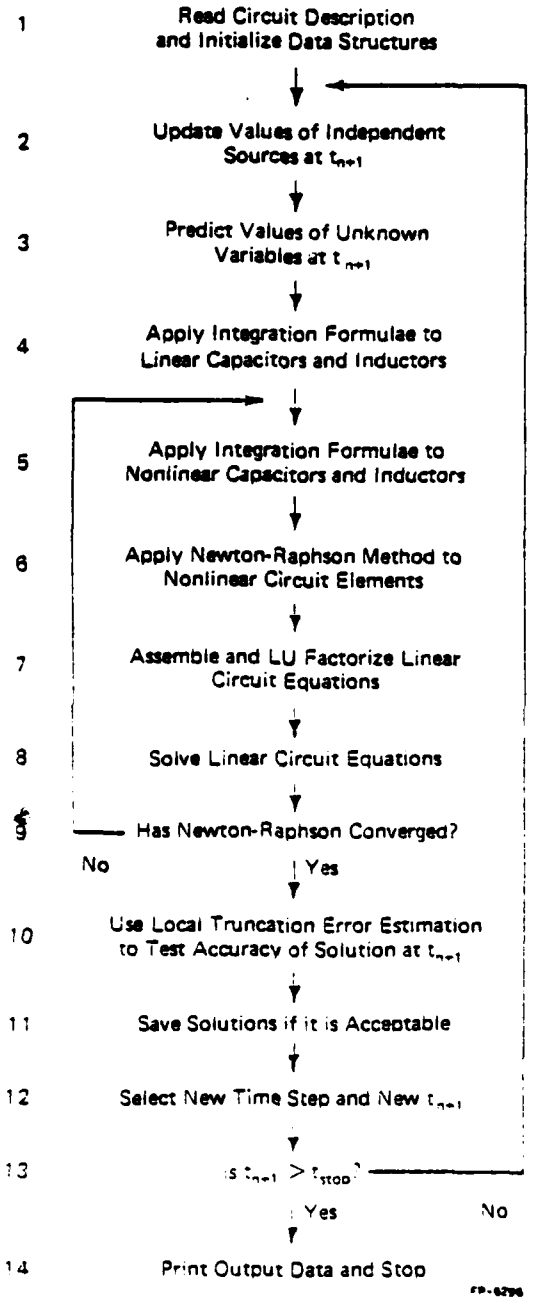


Fig. 1. Circuit simulator flow diagram for transient analysis [16].

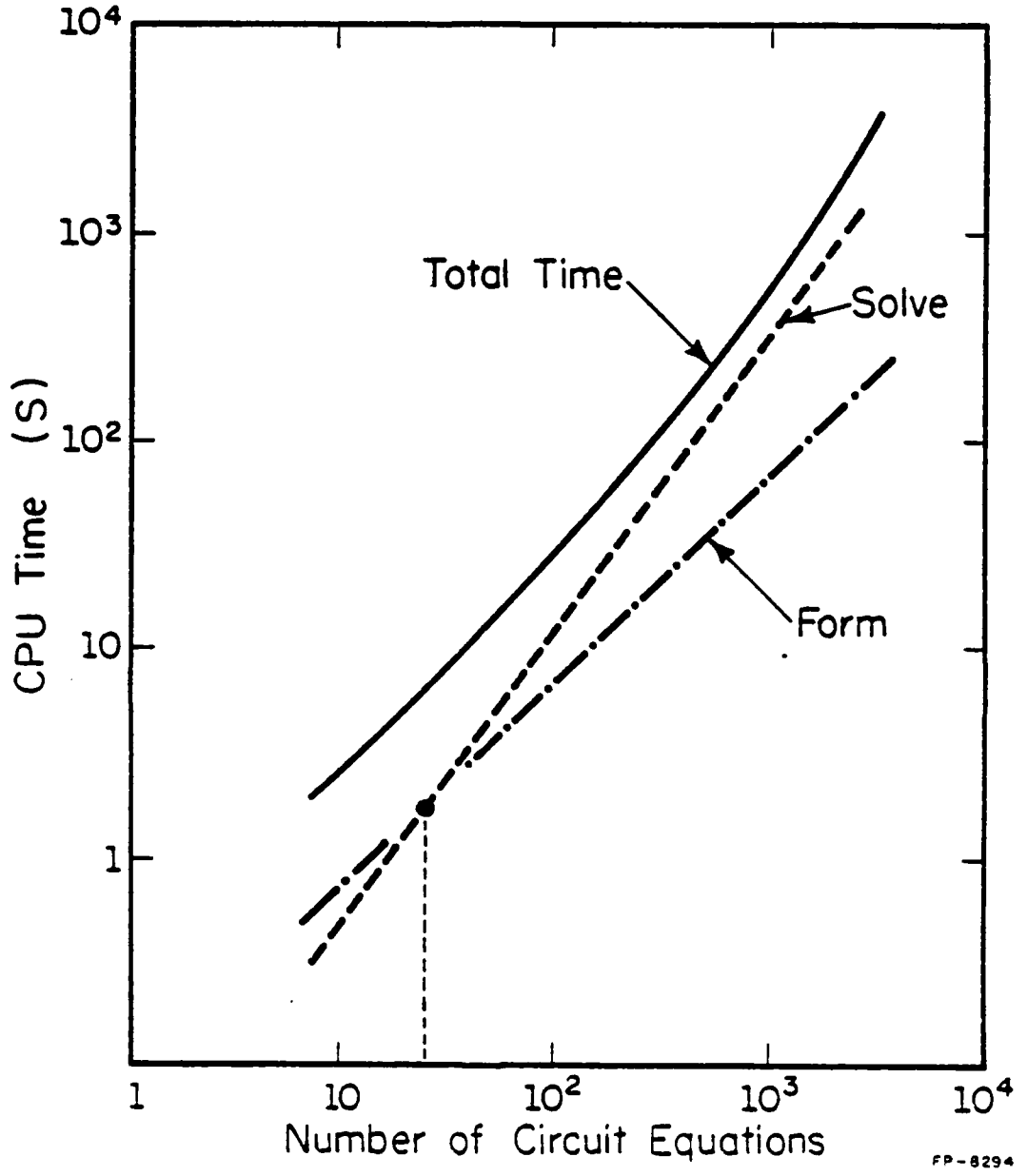


Fig. 2. Transient analysis time for circuits of increasing size [16].

size of the circuit grows, an increasing percentage of the time is spent in the SOLVE phase for all standard circuit simulators running on conventional computers. This thesis proposes a design of a special parallel processor to be used in performing the steps involved in SOLVE.

There are two methods for solving large systems of linear equations. One method is to use LU factorization, forward and backward substitutions on the entire matrix to arrive at the solutions in one pass. The other method is to use an indirect relaxation method where certain solutions are guessed at on the first iteration. The new solutions are then found and the process is repeated until the solutions converge. For most circuits, the fraction of nodes which are changing their voltage values at a given point in time decreases as the circuit size increases. For circuits containing over 500 MOSFETS, fewer than 20 percent of the node voltages change significantly over a simulation time step. Circuit simulators exploit this time sparsity or latency by using device-level or block-level bypass schemes. In a device-level bypass scheme, if the terminal voltages and branch currents of a circuit element do not change significantly from the previous iteration, its contributions to A and b in Ax=b are not re-evaluated, and the values computed during the previous iteration are used. In the block-level bypass, both the matrix element evaluation and the node solution steps are bypassed for each block of inactive connected circuit elements.

The indirect method is suitable for large MOS digital integrated circuits. However, for nonlinear analog circuits or digital circuits with floating capacitances, the solutions are not guaranteed to converge, or if they do converge, the convergence rate is slow [16]. Thus, the direct method is more suitable for analyzing these circuits.

The types of circuits that are analyzed are usually very large, with thousands of nodes. Although sparse matrix techniques have been found to be computationally efficient, partitioning the system matrix into a bordered-block diagonal form has been suggested as one way where parallel processing could be used to speed up circuit simulation [5]. Another approach could be the use of systolic and wavefront array processors [2] after possibly ordering the matrix in band form. However, the application of systolic arrays and wavefront array processors to the parallel solution of bordered block diagonal partitioned matrices has not yet been investigated. The purpose of this thesis is to propose a systolic wavefront array architecture for LU factorization of partitioned systems. The architecture is also compared to other parallel architectures for evaluation purposes. Using partitioning techniques, the circuit matrices may be ordered into bordered block diagonal form (BBDF) [5]. Bordered block diagonal matrices have the form shown in Fig. 3. The circuit can be divided into subcircuits which are connected together by a relatively small number of nodes. Some heuristic tearing schemes [14] on how the reordering can be done have been published. However, to obtain

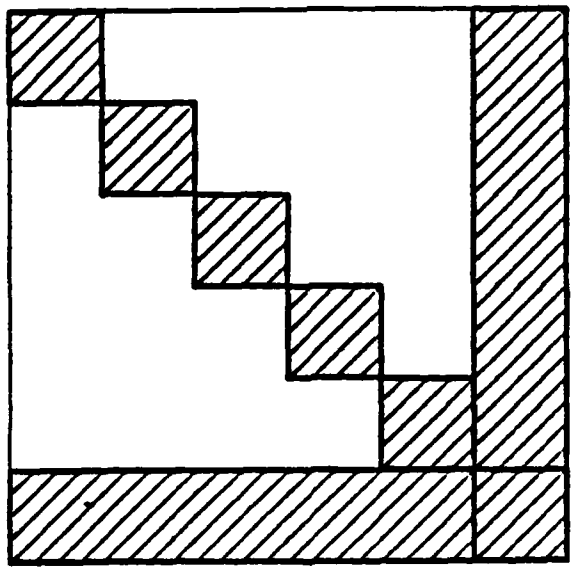


Fig. 3. Matrix in Bordered Block Diagonal Form (BBDF).

maximum concurrency, a nested form of the clustering algorithm is proposed in this thesis to reorder the nodes. In this approach, the entire circuit is first divided into two subcircuits in the BBDF form. Each of the two subcicuits is further divided into two subcircuits in BBDF in a recursive manner. This procedure continues until each subcircuit is approximately the minimum size allowed. LU factorization of the different subcircuits can then be carried out in parallel. However, the synchronization of the solutions at the interconnection level is a problem that needs careful consideration.

LU factorization involves repetitive computation on a large set of data. It involves a matrix multiplication-subtract operation which is a compute-bound problem. For matrices that are full, a great deal of concurrency can be achieved with cost-effectiveness on single-instruction-multiple-data (SIMD) machines because of the regular structure of the operations performed in the algorithm. However, as the matrix becomes sparse, it is very inefficient to use the same technique and, in case of parallel processing, the need for unstructured computations is too complicated for cost effective solutions on SIMD machines. This is because the LU factorization itself involves a great deal of data dependencies which lead to moderate parallelism.

The alternative architecture is multiple-instruction-multiple-data stream (MIMD). However, the need for synchronization of data, the amount of overheads allotted for arbitration between processors and/or memory conflicts usually degrade the performance. Thus, SIMD architecture is chosen for discussion.

The performance evaluation between the different types of architectures for LU factorization is based on the following main issues:

1. concurrency achieved by the machine
2. memory access and I/O time
3. synchronization of data
4. control complexity
5. hardware vs. software tradeoff of the algorithm
6. performance-cost effectiveness

The thesis is organized as follows. In Chapter 2, the steps involved in LU factorization are clearly defined and explained, followed by a study of general purpose and special purpose architectures for parallel processing. In Chapter 3, partitioned systems are solved in an implementation having both the characteristics of the systolic array and the wavefront method. Chapter 4 shows how the entire system is configured using the direct method of solving the matrices, whereas Chapter 5 discusses the configuration using an indirect Gauss-Seidel method. Chapter 6 is the conclusion of the thesis and suggests future direction of research. The block diagram of a CMOS processing element is given in the Appendix. Such a processing element can form a cell in a systolic array.

CHAPTER 2

BACKGROUND

2.1. Definition of LU Factorization

The problem that this paper concentrates on is the LU decomposition of large matrices which occurs in the solution of linear systems of equations. To solve the problem Ax = y, the matrix A is first decomposed into the product of a lower triangular (L) and an upper triangular (U) matrix, i.e., A = LU. Such an LU decomposition is unique if and only if A is strongly nonsingular [10]. Then a forward substitution step is performed where z is determined by solving Lz = y. The solution, x , can then be obtained by solving Ux = z. This is referred to as the direct method for solving linear systems of equations.

Crout's algorithm is one of the methods used in the decomposition. It is written in Pascal as follows:

```
For i:= 1 to SIZE do begin
   for j:= i+1 to SIZE do
      a[i,j] := a[i,j] / a[i,i];
   for k:= i+1 to SIZE do
      for w:= i+1 to SIZE do
         a[k,w] := a[k,w] - a[k,i] * a[i,w];
end;
```

where it is assumed that at every step, a[i,i] is not 0.

The efficiency of evaluating the inner loop in the above algorithm can be achieved through parallel processing of the machine and

through the choice of a good algorithm. The multiplication can be speeded up by special purpose hardware whereas the addition can also be speeded up by faster memories.

This algorithm can be implemented either in the software of general purpose machines or in the hardware of special purpose machines. The systolic array [4] and the wavefront array [2] processor are two different hardware implementations of the algorithm. The basic difference between the two architectures is that systolic arrays use a synchronous timing scheme whereas the wavefront array processors use a self-timed handshaking scheme. Their advantages and disadvantages will be compared in the following sections.

For both partitioned and non-partitioned matrices, there are two different classes of methods used for solving linear systems of equations. One of them is a direct method in which the results are computed in sequence in one pass through the hardware. The other method is an indirect method such as a relaxation-based Gauss-Seidel method. In this method, an initial guess is made for the values of the interconnection nodes in the first pass in order to solve for the values of the subcircuit nodes. The new values of the interconnection nodes are then compared with the previous guess. The whole procedure is repeated until the computed values converge. Latency [5] can be explored in this method so that only certain subcircuits need be updated after each iteration. If the solutions of these subcircuits are not within a specified tolerance, more iterations are needed. This procedure should reduce the complexity of evaluating partitioned

systems as will be seen in the next chapters; but is recommended to be used when the iteration process is expected to converge reasonably fast.

These two methods are discussed in the following chapters after a hardware scheme is chosen for the LU factorization array.

2.2. GENERAL PURPOSE vs. SPECIAL PURPOSE ARCHITECTURE

2.2.1. General Purpose Architecture

The main advantage of general purpose machines is the flexibility they offer. Different operations, algorithms, sizes of matrices and sparsities of matrices can be evaluated by changing the software program. If a certain algorithm is chosen, the LU factors of different matrices can be computed with only very slight modifications, e.g., changing certain parameters or adding several subroutines to handle sparsity. However, performance is sacrificed for this flexibility. A great deal of buffering, I/O and memory access and execution overheads result. A more complicated instruction set must be implemented in general purpose machines for various applications. Additional time is needed to decode these instructions and extra hardware and software must be incorporated as well to implement these instructions. There is also the data-routing problem to get the sequential data into and out of parallel arrays. This operation requires complex address calculations which are expensive on general purpose machines.

Another disadvantage is that with the fixed word length in the machine, rounding errors in algebraic processes, if not properly controlled, may lead to unreliable solutions. Special purpose machines can be designed with very good numerical control for these scientific computations. This can be achieved with special hardware for pivoting [6] so that the denominators of the divisions during LU factorization are always the largest numbers along the columns. This is usually done in the software of general purpose machine and is a time-consuming process.

The fact that fast general purpose machines are complicated to design and are usually very expensive calls for the design of special purpose machines that can be connected to an existing and relatively inexpensive host machine for fast numerical computation.

2.2.2. Special Purpose Architecture

The hardware cost and size of both special and general purpose machines are relatively insignificant compared to the software cost. However, the design cost of special purpose processors is usually much less than general purpose processors. A special purpose machine usually consists of simple processors of the same kind connected by a network of local and regular interconnects. Extensive concurrency can be achieved if the algorithm is designed to introduce high degrees of pipelining and multiprocessing. Data can be routed to the appropriate processor directly from memory to minimize the memory access time, which is the bottleneck of most algorithms. Thus, multiple computations can be performed per I/O access.

The bottleneck of using special purpose mesh connected processors is the time wasted in interprocessor data movement between the special purpose processors and the host. However, the computation can usually be done faster than general purpose machines. Less overheads are involved in decoding instructions and buffering because the architecture is geared only towards the computation needed.

One disadvantage of special purpose machines is their inflexibility. If a different faster algorithm for LU factorization is needed, the whole machine has to be redesigned and rewired. Also, in special cases, for example, sparse matrices which cannot be ordered in the block diagonal form may require complicated hardware circuitry in the special purpose machine as opposed to changing the software programs in the general purpose machine. Thus, there is a hardware versus software tradeoff between the two implementations.

Because of the inflexibility of special purpose processors, a well designed algorithm is an important starting point. However, a good algorithm for VLSI implementation may not necessarily be one requiring minimal computation. VLSI implementation is preferred for special purpose processors to reduce difficulties in reliability, performance and heat dissipation that arise from many SSI and MSI components. VLSI technology enables faster communication and more accurate system clocking due to the effect of scaling all components to several transistors and several metal and poly lines. VLSI also enables more modularity and programmability because of the cost-effectiveness of the design.

In summary, the criteria for special purpose machines are:

1. The design must be implemented by only a few different types of simple cells to cut down on design cost.
2. Its architecture must be based on a simple and regular data and control flow that can be connected by a network with local and regular interconnections.
3. It must fully utilize pipelining and multiprocessing. Several data streams can move at a constant velocity over fixed paths in the network, interacting at cells where they meet.
4. A large number of cells are active at one time so that computation speed can keep up with data rate.
5. The simple cells are identical and connected in a regular fashion to increase modularity and extensibility for different matrices.

Considering the above advantages and disadvantages, two different multiprocessor arrays, the systolic array and the wavefront array processors, are compared for their effectiveness in parallel processing.

2.3. Special Purpose Architecture

2.3.1. Systolic Array Processors

The systolic array is chosen as the special purpose machine for discussion because it satisfies all the criteria listed for performance/cost effective special purpose machines. Also, systolic arrays facilitate VLSI implementation by encouraging modular design

and local communication whereas many other special purpose machines that have been proposed [7,10] usually require LSI/MSI parts with massive controls and complicated hardware.

The term "systole" means a rhythmically recurrent contraction, especially the contraction of the heart by which the blood is forced onward and circulation kept up. This is analogous to the systolic array idea.

A systolic array processor consists of an array of simple processors connected together locally with inputs and outputs connected to a bus which is controlled by a host computer. The systolic system rhythmically computes and passes data through the system. All the inputs in the systolic cell move to the adjacent processor of the cell during each beat. The computation is then done rhythmically (i.e., depending on the beat) and the result is shifted as input to the next cell in the next beat. This is the basic data flow of the architecture.

According to S.Y.Kung [13], a systolic array must have the following properties:

1. Synchrony: The data transfer and computations must be controlled by a global clock.
2. Regularity: The array must consist of modular processing elements with regular and spatially local interconnections. Moreover, the network must be extendable.
3. Temporal Locality: At least one unit time delay must be allotted so that signal transactions from one node to the next can be

completed.

4. Linear-rate Pipelinability: The array must achieve an O(n) speedup, where n is the number of processing elements.

S.Y. Kung [13] has also devised a systematic way of systolizing an algorithm. A signal flow graph (SFG) is constructed to represent the input and output relationships of an algorithm. Then the basic operation module is selected. After that, two cut-set (temporal) localization procedures are applied:

Rule (i): Time-Scaling: all delays may be scaled by a single positive integer.

Rule (ii): Delay-Transfer: advance k time units on all the outbound edges and delay k time units on the inbound edges, and vice versa.

In the final step, combine the delay of the operation modules with the module operation to form a basic systolic element. All the extra delays will be modeled as pure delays without operations.

The signal flow graph for LU decomposition is shown in Fig. 4. Its corresponding systolic array is shown in Fig. 5. The array is organized like a honeycomb in a rhombus shape. The following functions are performed by each cell:

$$Aout = Ain - Lin * Uin$$

$$Uout = Uin \text{ or } Ain \text{ (at the boundary)}$$

$$Lout = Lin \text{ or } Ain/Uin \text{ (at the boundary)}$$

The number of cells needed in this topology is equal to the number of elements in a full matrix, e.g., for a full 4 x 4 matrix, 16 cells are needed. For a banded matrix, the number of cells needed is the

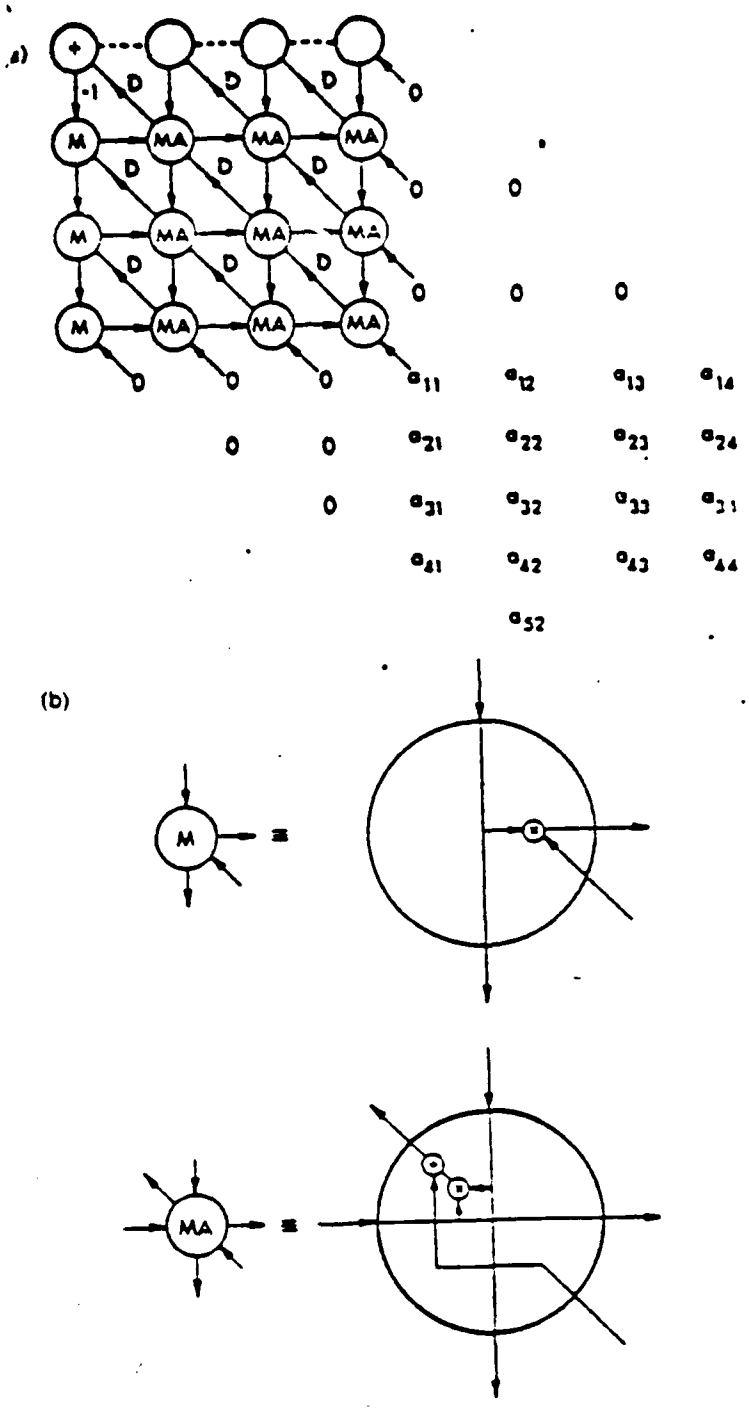


Fig. 4. (a) An SFG for LU decomposition. (b) The detailed diagram of the processing nodes [14].

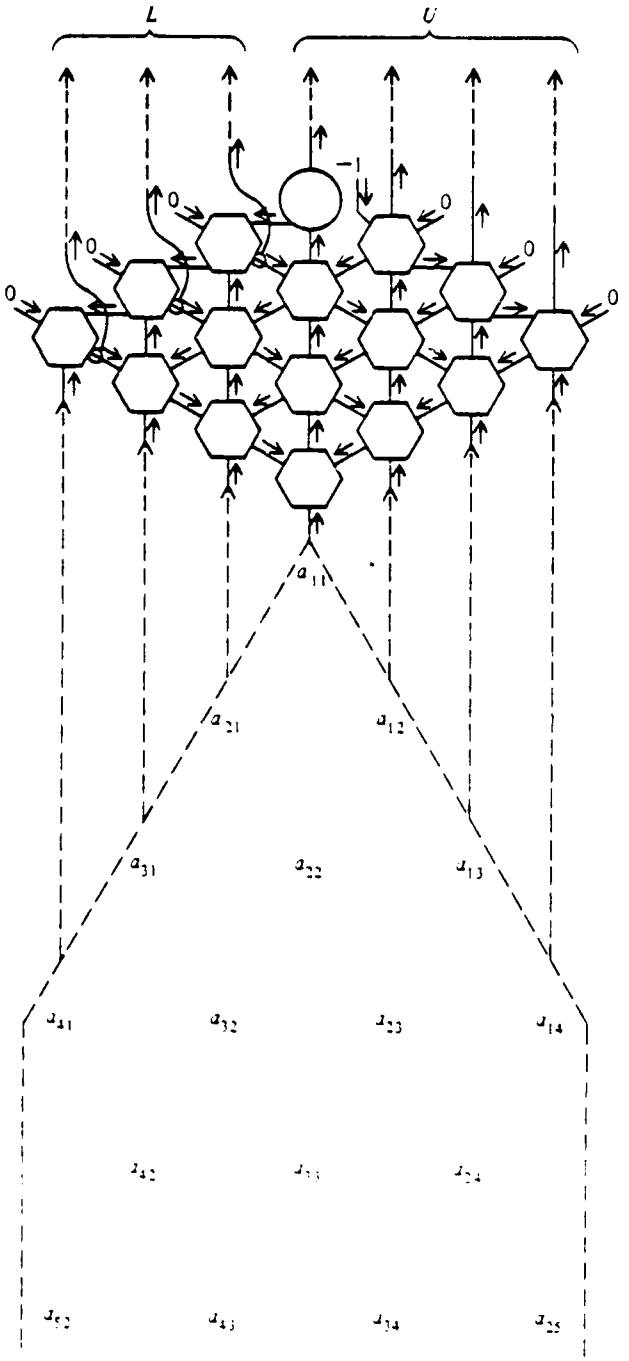


Fig. 5. The hex-connected processor array for pipelining the LU-decomposition of a 4 x 4 band matrix.

width of the first row times the height of the first column of the matrix. Only 1/3 of the cells are performing computation at each clock cycle while all of them are actively shifting data of some sort.

Because it is difficult to control the pipeline of systolic arrays, Kai Hwang [6] has proposed a rectangular implementation that uses n cells less for an n x n matrix but computes the LU factors faster than the above method. The speed is accomplished by utilizing more cells for computation at any given time. This method is different from the systolic array because interface latches are used to store intermediate results instead of using registers in cells. As a result, synchronization of data is very crucial in systolic arrays whereas data can be stored in the latches in the Hwang method. The interconnections and inputs of this topology are as shown in Fig. 6. The number of cells needed by the 4 x 4 full matrix is 12, which is less than the regular systolic array.

2.3.1.1. Advantages of Systolic Arrays

The systolic cell maximizes the use of input data fetched from memory with modest I/O bandwidths for outside communication with the host. This is done by inputting data at the appropriate cell at a regular rate and then shifting the data to the appropriate processor for computation. Thus, no multiple memory access of the same data is needed and no additional complex address calculations are required to retrieve data. When the memory speed is larger than the cell speed, two-dimensional systolic arrays are used because at each cell cycle,

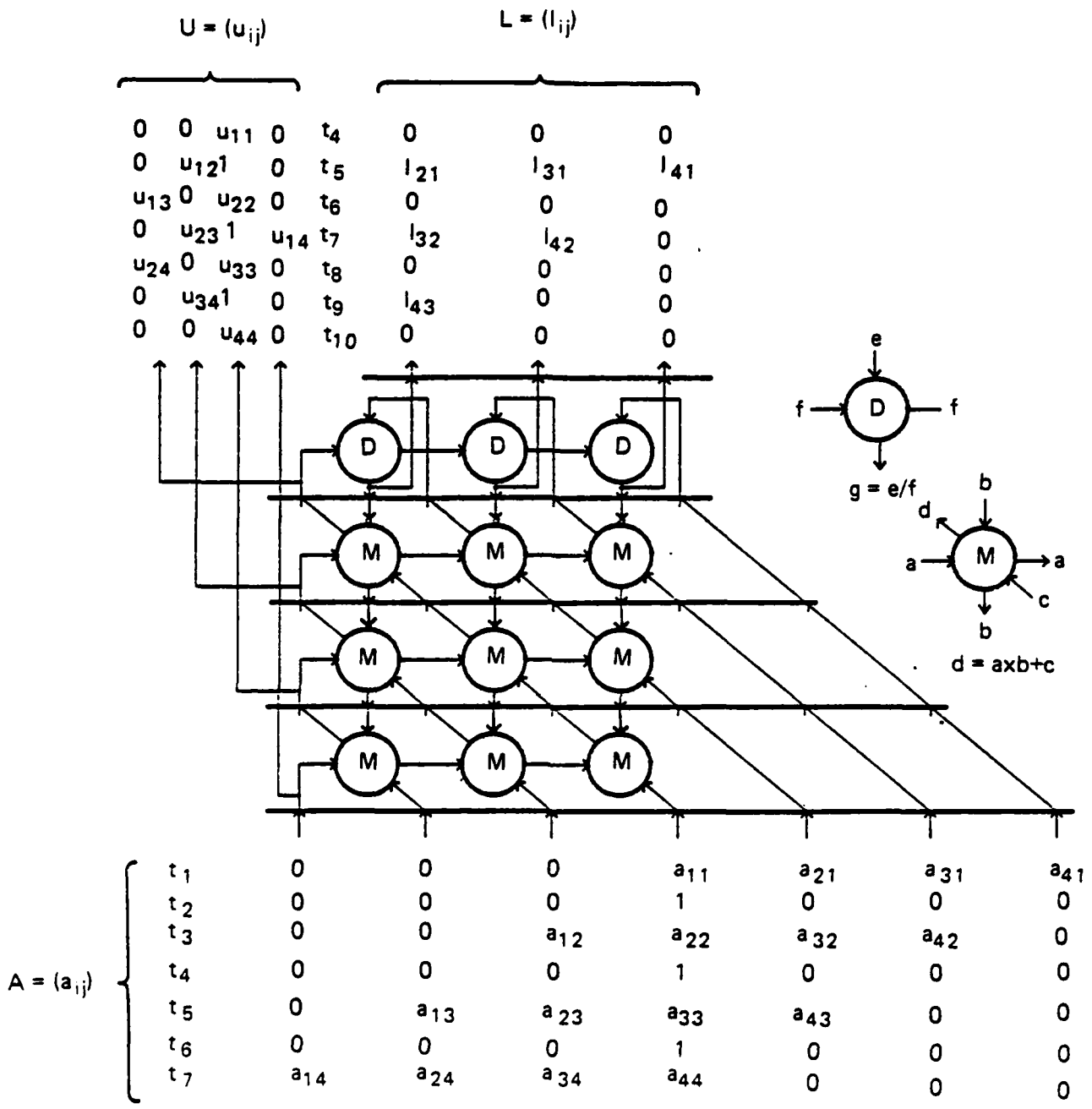


Fig. 6. VLSI Pipeline for LU decomposition using the Hwang Implementation [6].

all the I/O ports on array boundaries can be input or output data to or from memory. Memory bandwidth can be fully utilized and substantially reduces processor-memory traffic (memory access bottleneck) in comparison to that required in other architectures.

The algorithm on which the hardware of the systolic array is based fully exploits parallel-pipelined processing and speeds up compute-bound computations. Multiprocessing many results in parallel is achieved by converting the output (Aout) as input (Ain) to the next cell, and computation of each single result within the cell is also pipelined. Thus the systolic cell is a two-stage pipeline. System cycle time is the time of a stage of a cell and not the whole array of cells at cycle time. Very fast computation, up to 200 Mflops (million floating point operations per second) [3], can be achieved.

The cells are also simple and identical. The cell can basically consist of shift registers or latches, a multiplier, a reciprocator and an accumulator. A local memory may also be included to store intermediate operands and results. Data and control flows are simple and regular as shown in the Kai Hwang [6] systolic array.

The design of the systolic cell is completely modular and expandable with no difficult synchronization. Additional cells can be connected together in the same topology for larger matrices. In fact, as the number of cells expands, system cost and performance increase proportionally if the size of the underlying problem is sufficiently large.

One other advantage of special purpose machines over general purpose machines is that the software overhead associated with operations, such as indexing, are totally eliminated.

2.3.1.2. Disadvantages of Systolic Arrays

Global synchronization, however, could be a disadvantage as the size of the array increases. While an asynchronous model incurs a fixed time delay overhead due to the handshaking process, the synchronous time delay is primarily due to the clock skew which increases with the size of the array. Even in an H-tree clock distribution [8] which maintains the same on-chip clock line length to each of the N x N modules, the clock skew grows at a rate of $O(N^3)$. In addition, the systolic arrays are rather inflexible to changes in the algorithm, for example, the partitioning of sparse matrices into block diagonal form for higher concurrency may lead to interconnection hardware that is extremely complex to design, whereas solving the huge sparse matrices using unpartitioned matrix systolic operation is a waste of processor and waste of computation time as most elements are zero elements unless the matrix is banded. Even for banded matrices, there is a limit of 4n on execution time [13]. Interconnection of the submatrices of sparse matrices , if efficiently solved in hardware, may lead to special purpose machines with very high throughput. Several schemes that are not systolic in nature have been proposed. They are the LU unstructured sparse matrix machine by Pottle [7], partitioned matrix by Hwang and Cheng [10] and the Blossom System by Sangiovanni-Vincentelli [1].

2.3.1.3. Timing Analysis and Number of Processors

Four schemes used for LU factorization are considered:

1. Regular LU banded matrix

   Decomposition: $T = 3n + \min(r,s)$; $N = rs$

   Triangularization: $T = 3n$; $N = (n^2 + n)/2$

2. LU unstructured sparse matrix by Pottle

   $T = nl$; $N >= dmax$

3. Partitioned matrix algorithm by Hwang and Cheng

   $T = O(n)$ or $O(n/m)$; $N = O(n)$ or $O(Nm)$

4. Blossom by Sangiovanni-Vincenteli

   $T = O(n/x)$ for full matrices, else $O(Nwh)$; $N = x$

where,

   T is the time complexity

   N is the number of processors

   n is the maximum size of the matrix

   r+s-1 is the bandwidth of a banded matrix

   r is the width of the first row of the banded matrix

   s is the length of the first column of the banded matrix

   l is the mean degree of columns of a sparse matrix

   m is submatrix size

   w is the original border width

   h is a function of x and m

   dmax is the maximum number of nonzeros in any row of the operand

The hardware suggested by Pottle [7] is difficult to design and involves a great deal of control signals and local storage. More-

over, there is not enough information on the processor array used in the Blossom system [1]. However, the systolic array and the Hwang [6] scheme are easy to control and to design. The systolic array is more suitable for banded matrices while the Hwang [6] scheme is better suited for full matrices. However, in our applications, the matrix is assumed to be partitioned into submatrices so that each submatrix is nearly a full matrix.

2.3.2. Wavefront Array Processor Architecture

The wavefront array processor (WAP) is configured in a square array of NxN processing elements with regular and local interconnections (Fig.7). The computing network serves as a (data) wave-propagating medium. The computational sequence starts with one element and propagates through the processor array, closely resembling a physical wave phenomenon. A second wavefront can then be pipelined immediately after the first one propagates. Also, wavefronts of two successive recursions of the software loop will never intersect (Huygen's Wavefront Principle) [13] because the processors executing the recursions at any given instant will be different, thus avoiding any contention problems.

According to S.Y. Kung [13], a wavefront array is a computing network which possesses the following properties:

1. Self-Timed, Data-Driven Computation: The network is data-driven, i.e., the computation is fired as soon as the data arrives; thus, no global clock is needed.

2. Modularity and Local Interconnection: This property is the same

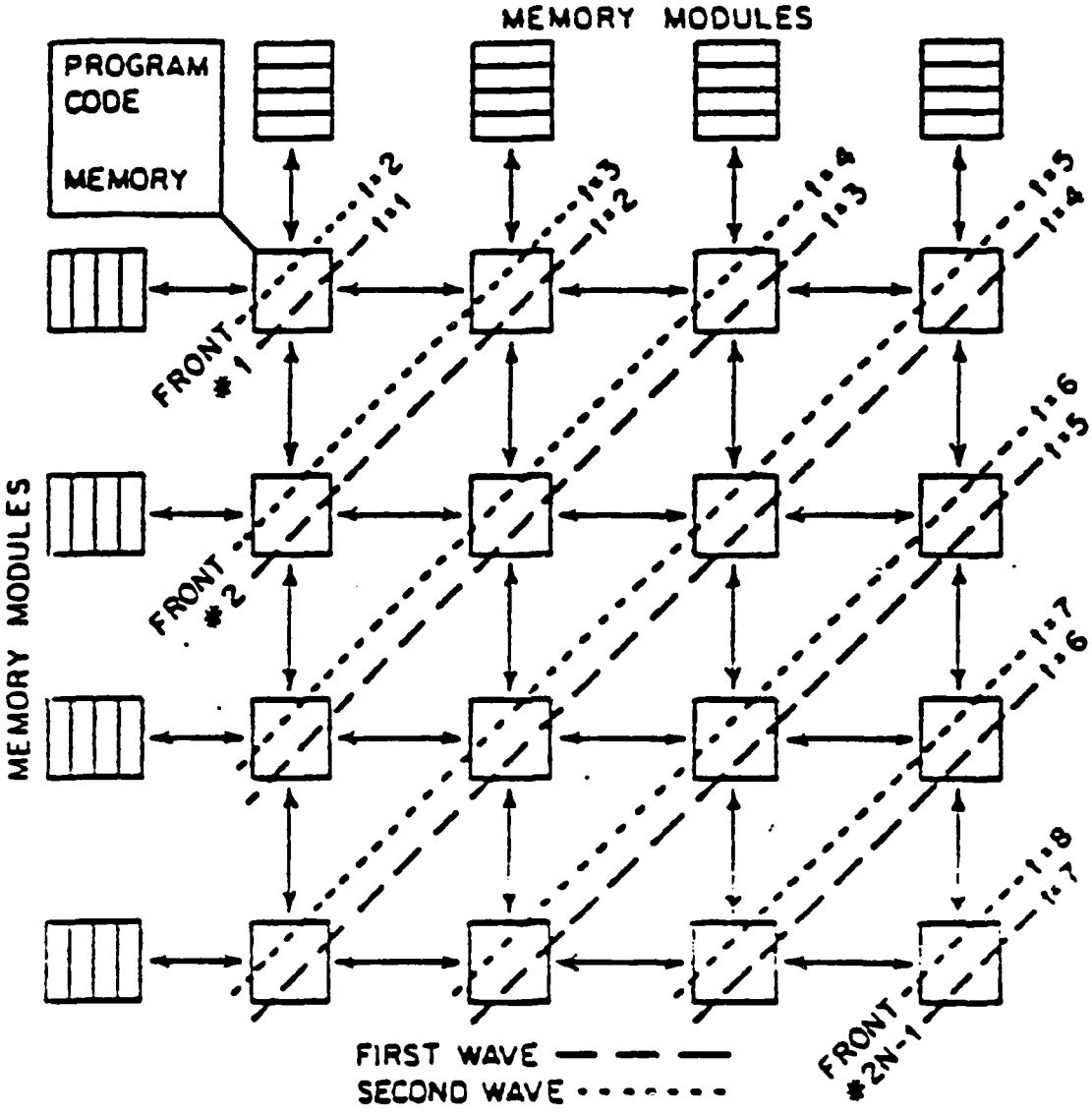


Fig. 7. The WAP configuration [2].

as the systolic array, except that the wavefront array can be extended indefinitely without global synchronization problems.

3. O(n) Speedup and Pipelinability: similar to the systolic array.

Thus, the only major difference between the wavefront and systolic array is the data-driven property. Therefore, temporal locality is no longer needed in the wavefront array and thus the wavefront array is faster and easier to program.

To perform LU factorization, each recursion consists of a main wavefront and an alternate wavefront. The main wavefront computes Li at the (*,i) processors and sends it left to form the columns of L in the left memory modules. It also computes Ui at the (1,*) processors and sends it up to form the rows of U. The '*' means all the elements in a particular row or column indicated. In the interior processors, Ai = A(i-1) - Li * Ui is computed. The main wavefront must also send the new Ai's up. The alternate wavefront sends Aij down and left. After the first pair of wavefronts is initiated, the second pair can be pipelined immediately.

2.3.2.1. Advantages and Disadvantages of Wavefront Array Processors

The WAP has the advantage of the systolic array in the fact that it is very modular and with local interconnection. However, a wavefront architecture can also provide asynchronous waiting capability and consequently can cope with timing uncertainties such as local clocking, random delay in communications and fluctuations of comput-

ing times. Thus, the notion lends itself to a (asynchronous) data-flow computing structure that conforms well with the constraints of VLSI.

The wavefront notion drastically reduces the complexity in the description of parallel algorithms. The mechanism provided for this description is a special-purpose, wavefront oriented language. Rather than requiring a program for each processor in the array, this language allows the programmer to address an entire front of processors. This solves the problem of programmability and extensibility of systolic arrays.

The WAP shares a key concept with data-flow machines: the arrival of data fires each processing element (PE), which subsequently sends relevant data to the next PE. The WAP can be regarded as a homogeneous data-flow processing element. The 'WAIT' for data feature, provided by handshaking, allows for the globally asynchronous operation of processors, i.e., there is no need for global synchronization. Scheduling and synchronization are built at the hardware level. These qualities make the WAP very appealing for VLSI implementation as well. Figs.8 and 9 show the difference between synchronous and asynchronous handshaking. Fig. 10 shows the delay modules for two adjacent asynchronous modules. The loop delay, dA, is the time delay between servicing successive data bits for the asynchronous architecture. It is given by [8]:

$$dA = 2dL + dF + dP$$

where dL = propagation delay of the combinational logic

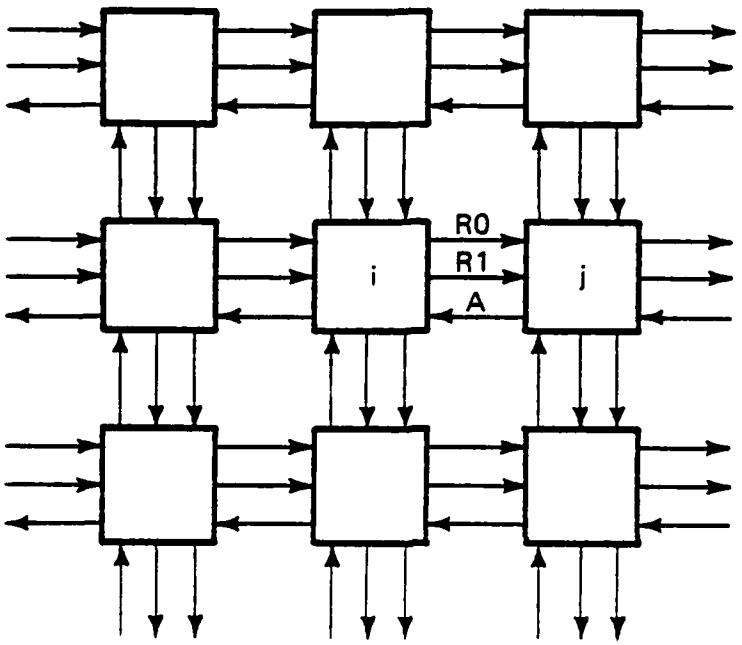


Fig. 8. Crossbar interconnection network using asynchronous modules [8]. R0 is used for logic 0 data and R1 for logic 1 data. A is the acknowledge signal.

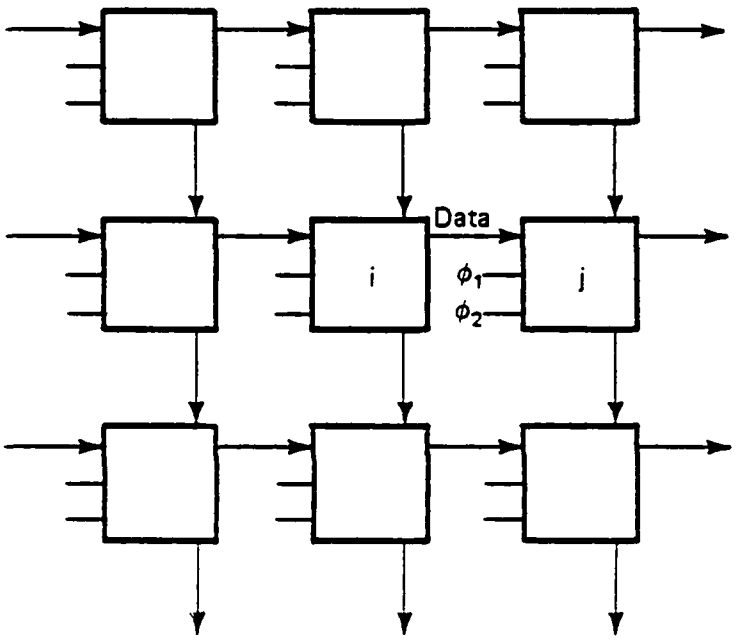


Fig. 9. Crossbar interconection network using synchronous modules [8].

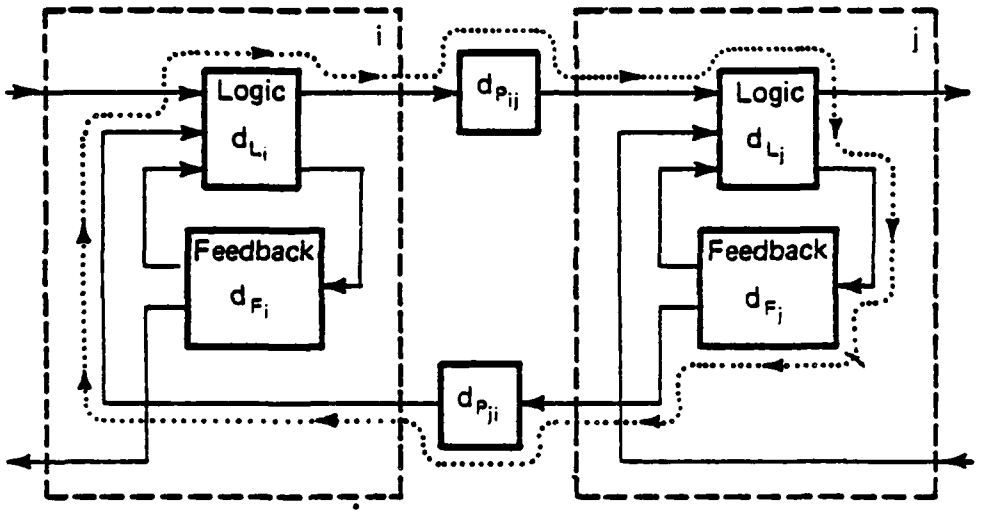


Fig. 10. Delay module for two adjacent asynchronous modules [8].

dF = propagation delay of the feedback path

dP = propagation delay along the request path from module i to module j

However, the synchronous delay module as shown in Fig. 11 is given by:

$$dS = dL + 2dM + dP + dC$$

where dM = propagation delay of the memory elements

and dC = dci - dcj = clock skew

Thus for large dP, the synchronous system data rate is higher than that of the asynchronous system whereas as the clock skew is increased, the asynchronous system is better.

In conclusion, the WAP is an optimal tradeoff between globally synchronized and dedicated systolic array and general purpose data-flow multiprocessor. The Hwang (rectangular) configuration of the systolic array is analogous to the WAP . The difference is that input data is synchronously fed into the array in the Hwang implementation whereas in the WAP, the network is data-driven. However, the asynchronous handshaking and the wavefront language based programming control can be implemented for this rectangular systolic array. Thus, the rectangular configuration is chosen for analysis of the LU factorization of both normal bordered block diagonal form (BBDF) and nested BBDF matrices. This thesis describes the design of a parallel processor array for LU factorization by modifying and combining the traditional systolic array, the Hwang implementation and the wave-front array processor. This modified design has the characteristic

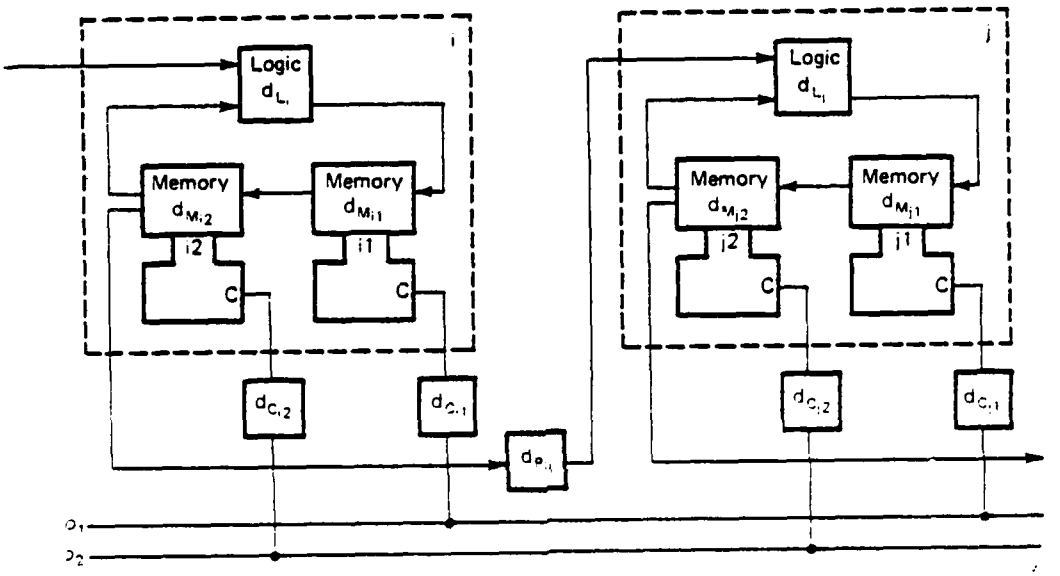


Fig. 11. Delay module for two adjacent synchronous modules [8].

of the systolic array because within each level of the processor array, the data are fed in a synchronous manner, each beat being controlled by a system clock. The Hwang implementation of the interface latches are used because more processing elements are then utilized for computation. In addition, the asynchronous handshaking scheme is very useful for taking care of the irregularity of the outputs from the previous processing level. The asynchronous handshaking only needs to take place at the cells located at the edge of the processor array where the input data are fed. The software program can generate an input data valid flag to start the wavefront of input data at each clock pulse. The Appendix gives the VLSI design of such a semi-synchronous processing element. It is semi-synchronous because it has both a system clock and asynchronous handshaking signals. The asynchronous handshaking is basically needed as the outputs of a certain level of processor array flows as inputs to the next level. This is because the processing elements may not produce the outputs in the pattern needed as inputs to the subsequent levels.

CHAPTER 3

ARCHITECTURE FOR LU FACTORIZATION OF PARTITIONED SYSTEMS

3.1. Characteristics of the Matrices

Most of the matrices in circuit simulation programs are large and sparse. Each node in the circuit is connected to a small subset of nodes. Thus, the circuits can usually be ordered into Bordered-Block Diagonal Form (BBDF) [5].

$$\begin{bmatrix} M1 & 0 & 0 & 0 & Y1c \\ 0 & M2 & 0 & 0 & Y2c \\ & \cdots & & & \cdots \\ 0 & 0 & 0 & Mk & Ykc \\ Yc1 & Yc2 & 0 & Yck & Ycc \end{bmatrix} \begin{bmatrix} X1 \\ X2 \\ \cdots \\ Xk \\ Vc \end{bmatrix} = \begin{bmatrix} Y1 \\ Y2 \\ \cdots \\ Yk \\ Ic \end{bmatrix}$$

Parallel processing can be utilized by finding the LU factors of all the subcircuits at the same time and then the results are automatically merged properly together to form the interconnection level. The following matrices are then factored in parallel:

$$\begin{bmatrix} M1 & Y1c \\ Yc1 & 0 \end{bmatrix} \quad \begin{bmatrix} M2 & Y2c \\ Yc2 & 0 \end{bmatrix} \quad \cdots\cdots \quad \begin{bmatrix} Mk & Ykc \\ Yck & 0 \end{bmatrix}$$

As discussed in the previous section, a systolic approach can be used in the architecture. Two implementations of the mesh connection are studied: the Hexagonal systolic array by Kung[4] and the

rectangular network by Hwang [6]. It is found, using hand simulations, that the Hwang [6] scheme takes advantage of maximum processor utilization as well as taking fewer clock cycles for processing the matrix. The hexagonal systolic array can also be described as a rectangular array. However, the difference between the Hwang [6] method and the regular systolic array is in the use of interface latches in the Hwang [6] method so that the timing is different between the two. A modified scheme of these two methods is studied in the next section.

The partitioned matrix is better suited for parallel processing than the banded matrix. It can be seen from a comparison of the time it takes a regular systolic array to compute the LU factors of a banded matrix and a BBDF matrix (Fig. 12). As p and q gets << n and as m gets larger, t(partitioned) < t(banded). Using the systolic array by H.T. Kung [4],

$$t(band) = 3n + \min(p,q) \sim 3n + p$$

Assume q is either larger or equal to p.

$$t(partitioned) < t(sub) + t(int)$$

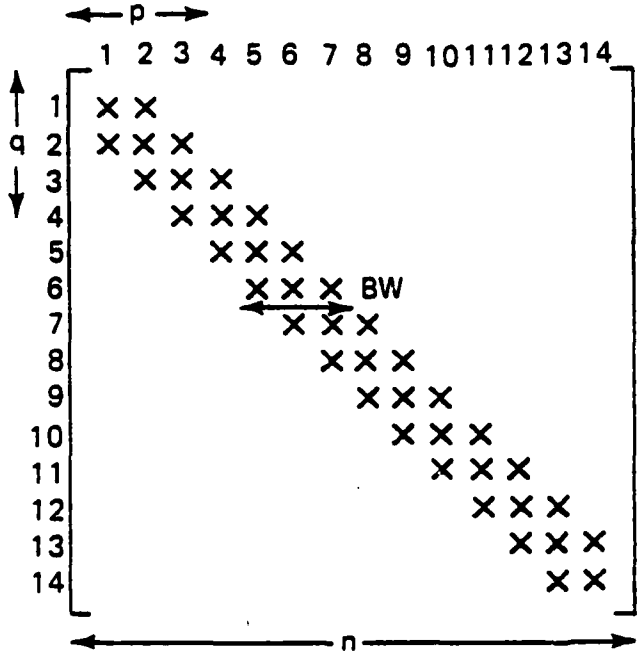
$$t(sub) = 4[p+n-mp]$$

$$t(int) <= 4(n-mp)$$

Assume summation level is done almost simultaneously. Also, assume a full matrix at the interconnection, whereas most of the time, the interconnection can be ordered as a band matrix.

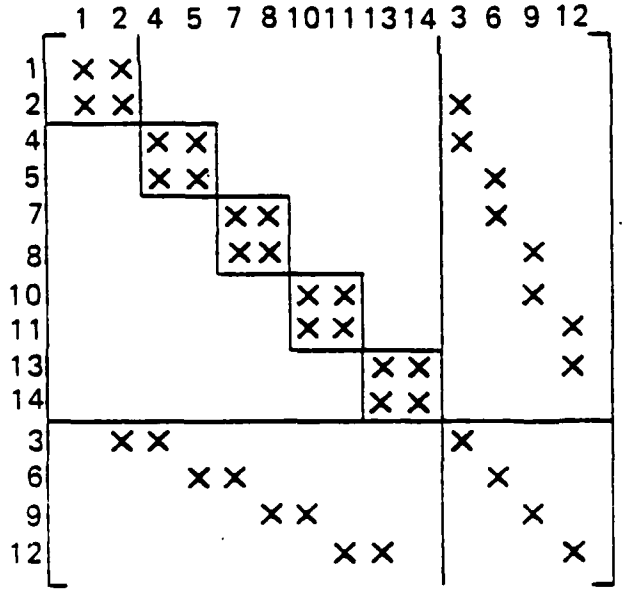
$$t(part) = 4[p+n-mp+n-mp] = 4[p+2n-2mp]$$

One of the problems is to find the optimal number (m) of subcircuits



Banded Matrix

$t_{band} = 3 \times 14 + 2$
$= 44$



Bordered Block Diagonal Matrix

Assuming Full Submatrices
$t_{sub} = 4 \times 6 = 24$
$t_{int} = 3 \times 4 + 1 = 13$

$t_{part} < 37$

FP-6290

Fig. 12. Comparison of the time taken to perform LU factorization on a banded matrix and a BBDF matrix using systolic arrays.

to divide the entire circuit.

For t(band) >= t(partitioned)

$$3n+p \geq 8n-8mp+4p$$

$$(8m-3)p \geq 5n$$

Thus for the partitioned matrix to be faster than the band matrix, make p and m large.

If the interconnection subcircuit is a band matrix, each border node depends only on w nodes (Fig. 13).

$$t(sub) = 4[p+w]$$

$$t(int) = 3[n-mp]+r$$

$$t(part) < t(sub) + t(int) = 4(p+w) + 3(n-mp) + r$$

$$t(band) = 3n+p$$

For t(band) >= t(part)

$$3n+p \geq 4(p+w) + 3(n-mp) + r$$

$$3p(m-1) \geq 4w + r$$

For example, given r=1, p=2, w=1

$$m \geq -5/6+1 \quad \text{e.g. } m = 2,3,4,....$$

However, the problem with the normal BBDF form of matrices is that the interconnection matrix and the border can get very large and sparse. Then, the control of data from the subcircuits to the interconnection level can be very complicated. Thus, an algorithm for partitioning the matrix is investigated. The algorithm uses a clustering procedure to obtain a 'nested' partitioned matrix form. The reason that the clustering algorithm can be useful is that most circuits considered are only connected to a few other nodes. So one can

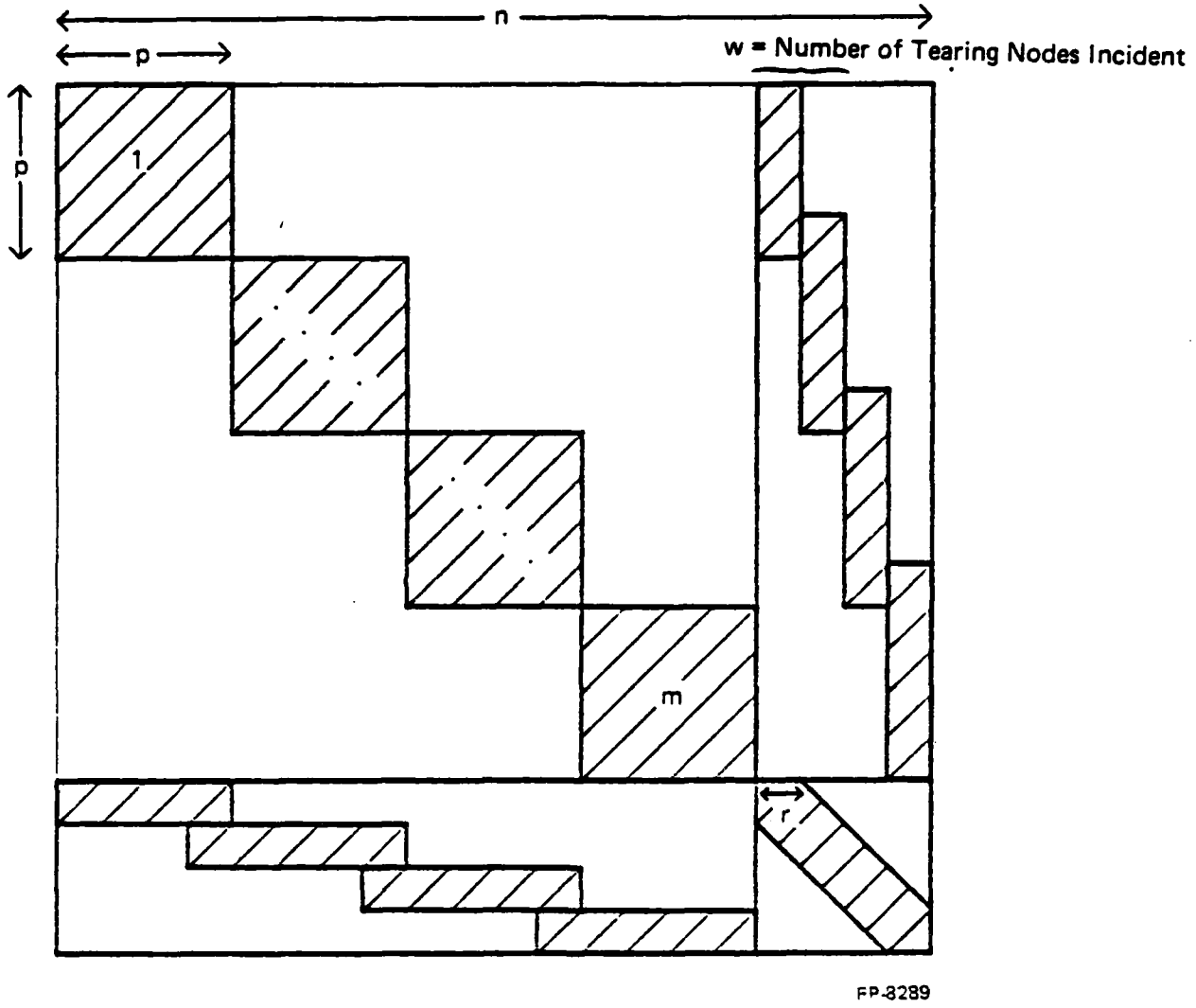


Fig. 13. A BBDF matrix with sparse interconections.

always divide the whole circuit into subcircuits which are connected only through a relatively small number of interconnection nodes.

The matrix is divided into BBDF in a nested manner, starting with 2 submatrices, then 2^2 matrices up to 2^d matrices. Each submatrix is also in BBDF. This method is used to simplify the interconnection level logic. With the original BBDF, the border can get very large but sparse. To utilize the sparsity maximally requires very complicated and sometimes irregular timing controls. However, using a nested algorithm, even though more mesh layers may be needed, the synchronization of data is very simple.

3.2. The Nested Clustering Algorithm

The problem with any heuristic clustering algorithm is the uneven size of the subcircuits. This can be remedied by patching the circuits with 1's on the diagonal and zeros everywhere on the extra rows and columns (Fig.14). The tearing nodes chosen may not be the minimum and may not give the most identical number of clusters since a certain small number, ε , has to be chosen around the vicinity of nmax allowed for each level of nesting. The clustering algorithm is performed by the following procedure, and is a modification of the algorithm given in [14]:

1. Choose initial iterating node [IS(1)] with minimum degree.
2. Store in adjacency set [AS(1)] all nodes that are adjacent to the node in IS(1).
3. Place the cardinality of AS(1) in CN(1).

$$\begin{bmatrix} a_{11} & a_{12} & 0 & 0 \\ a_{21} & a_{22} & 0 & 0 \\ 0 & a_{32} & a_{33} & 0 \\ 0 & 0 & a_{43} & a_{44} \end{bmatrix} \longrightarrow \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & a_{11} & a_{12} & 0 & 0 \\ 0 & a_{21} & a_{22} & 0 & 0 \\ 0 & 0 & a_{32} & a_{33} & 0 \\ 0 & 0 & 0 & a_{43} & a_{44} \end{bmatrix}$$

Fig. 14. Changing the size of matrices by adding 1's and 0's.

4. Let i=1.

5. If CN(i)=0, check if any node is not covered yet. If all the nodes are considered, stop; otherwise, continue.

6. Choose next iterating node, n(i+1), from AS(i) and place it in IS(i+1) by a greedy strategy with minimum CN(i+1).

7. Update AS(i+1) from AS(i) by deleting the node n(i+1) and adding all new nodes that are adjacent to n(i+1) that are not already in AS(i) or in the union of the sets IS(j) where j equals 1 to i.

8. Put CN(i+1) = |AS(i+1)|.

9. let i= i+1.

10. Go to step 5 until $\lceil n/2 \rceil$ nodes have been selected. Then choose the node, k, between nmax $\pm$ s that gives the minimum CN. s is the smallest integer that can give the minimum CN. All nodes 1 -> k form a cluster with all the nodes in the adjacency set as tearing nodes.

11. Now for the first cluster, divide the cluster in half again as in step 10. Repeat step 11 until the clusters are at a minimum size.

12. Also, delete the first cluster and its tearing nodes and continue steps (6) to (10) for the second cluster. Repeat step (11) until the clusters are at a minimum size.

Refer to Figs. 15a, 15b, and 16 for an illustration of the algorithm. The clustering algorithm is a software solution to organizing the interconnection nodes in groups before processing through the processor array. Using the nested BBDF, more parallel processing can be done with simpler interconnection, as is shown next.

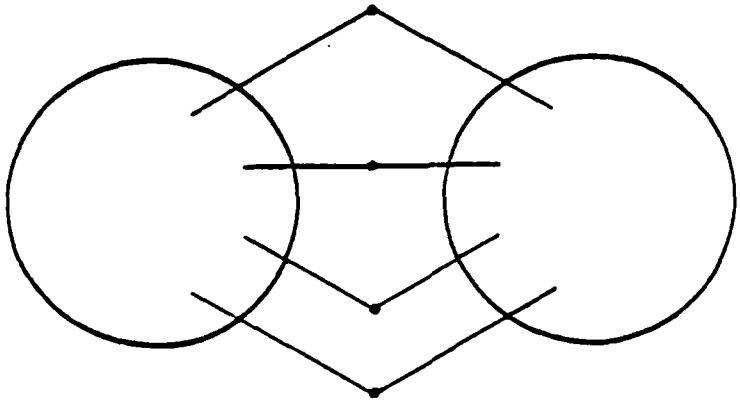


Fig. 15a. The nested clustering algorithm splits the entire circuit in half in the first step.

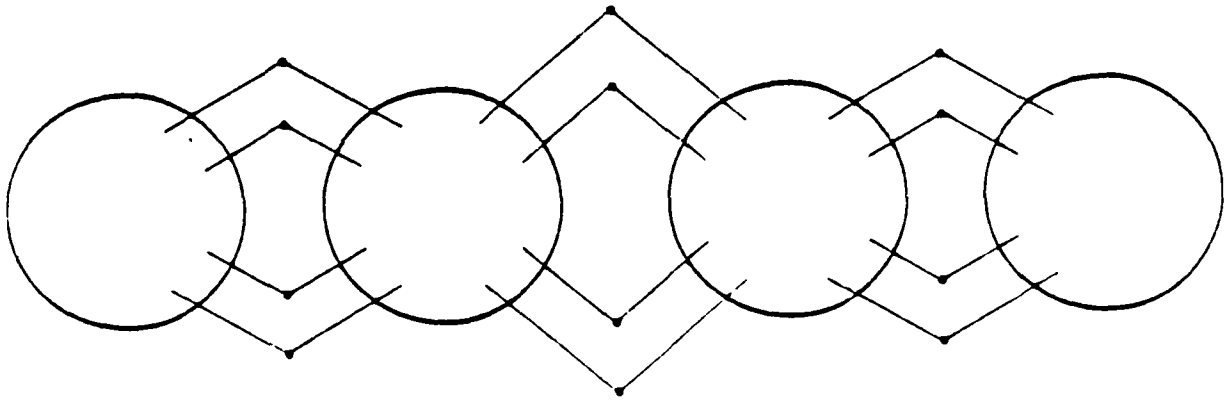
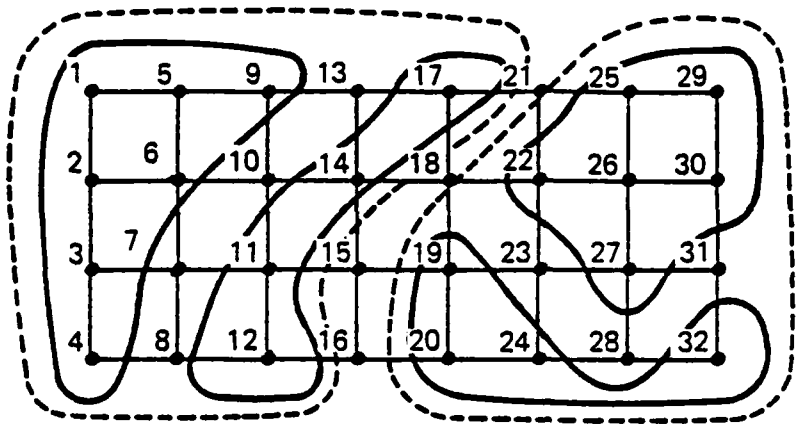


Fig. 15b. Then each individual cluster is furthur split in half during the next recursion.



No.	Is	As	Cn	
1	1	2, 5	2	
2	2	5, 3, 6	3	
3	5	3, 6, 9	3	
4	3	6, 9, 4, 7	4	
5	6	9, 4, 7, 10	4	
6	9	4, 7, 10, 13	4	
7	4	7, 10, 13, 8	4	Bottleneck Nodes [7,10,13,8]
8	7	10, 13, 8, 11	4	
9	10	13, 8, 11,14	4	
10	13	8, 11,14,17	4	
11	8	11, 14, 17, 12	4	
12	11	14, 17, 12, 15	4	
13	14	17, 12, 15, 18	4	
14	17	12, 15, 18, 21	4	
15	12	15, 18, 21, 16	4	Bottleneck Nodes [15,18,21,16]
16	25	26, 29	2	
17	26	29, 22, 30, 27	4	
18	29	22, 30, 27	3	
19	22	30, 27, 23	3	
20	30	27, 23, 31	3	
21	27	23, 31, 28	3	Bottleneck Nodes [23,31,28]
22	23	31, 28, 24, 19	4	
23	31	28, 24, 19, 32	4	
24	28	24, 19, 32	3	
25	24	19, 32 20	3	
26	19	32, 20	2	
27	32	20	1	
28	20		0	

Fig. 16. Example of the nested clustering algorithm being used on a grid circuit.

3.3. Design of the Modified Systolic Array

The hexagonal systolic array is orgranized in a honeycomb shaped mesh. Its topology can actually be implemented in a rectangle. The data flow of such a rectangular systolic array will still be the same as the hexagonal topology. The four diagrams shown in Fig.17 [13] illustrate the data flow of such a systolic array. Input data is fed along the diagonals of the matrix. The L factors are then taken from the left-hand edge of the network while the U factors are taken from the right-hand edge of the network. For the given 30 x 30 nested BBDF matrix, the input timings are shown in Fig.18. The numbers indicate the time that the input data at the corresponding position are fed, e.g., element (1,1) is fed at time t_1. The output timings are shown in Fig. 19. The numbers indicate the time that the output comes out of the processor array.

The inputs are fed along the diagonal of the matrix with the diagonal element first, followed by the rest of the elements along the diagonal. Zeros are fed in at the location of the interconnection level at each of the subcircuits. After all the subcircuit data-independent elements are fed into the array, the rest of the zeros are fed in with all the diagonal elements at the same time. Note that the input data streams are separated from each other by a clock pulse. At the interconnection location, the partial sums, $\sum$ Lik * Ukj are carried out.

To insure the correct timing, an asynchronous handshaking scheme is incorporated on top of the synchronous data sequencing. No

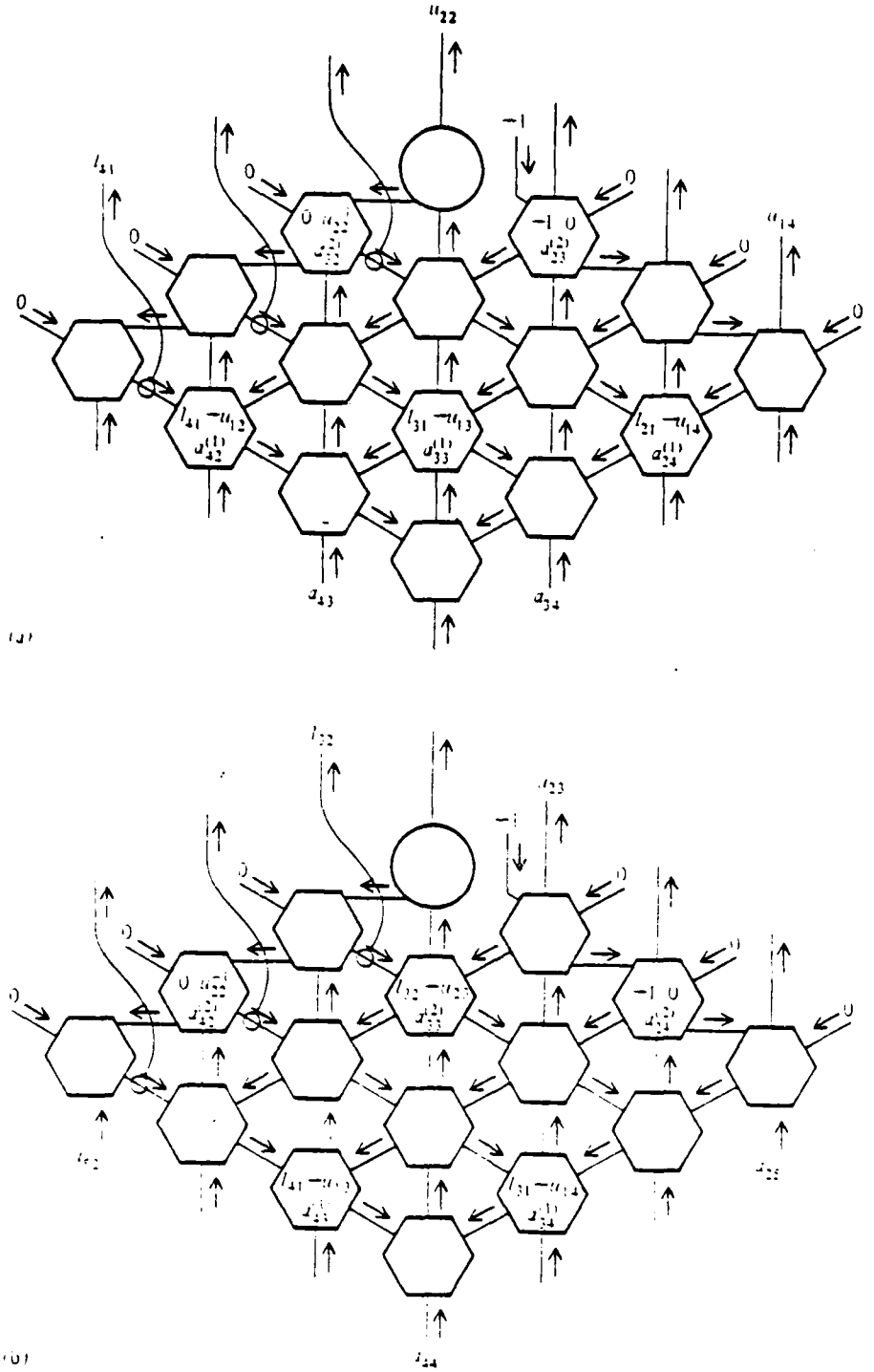


Fig. 17. Four steps during the LU-decomposition of the matrix shown in Fig. 5 [13].

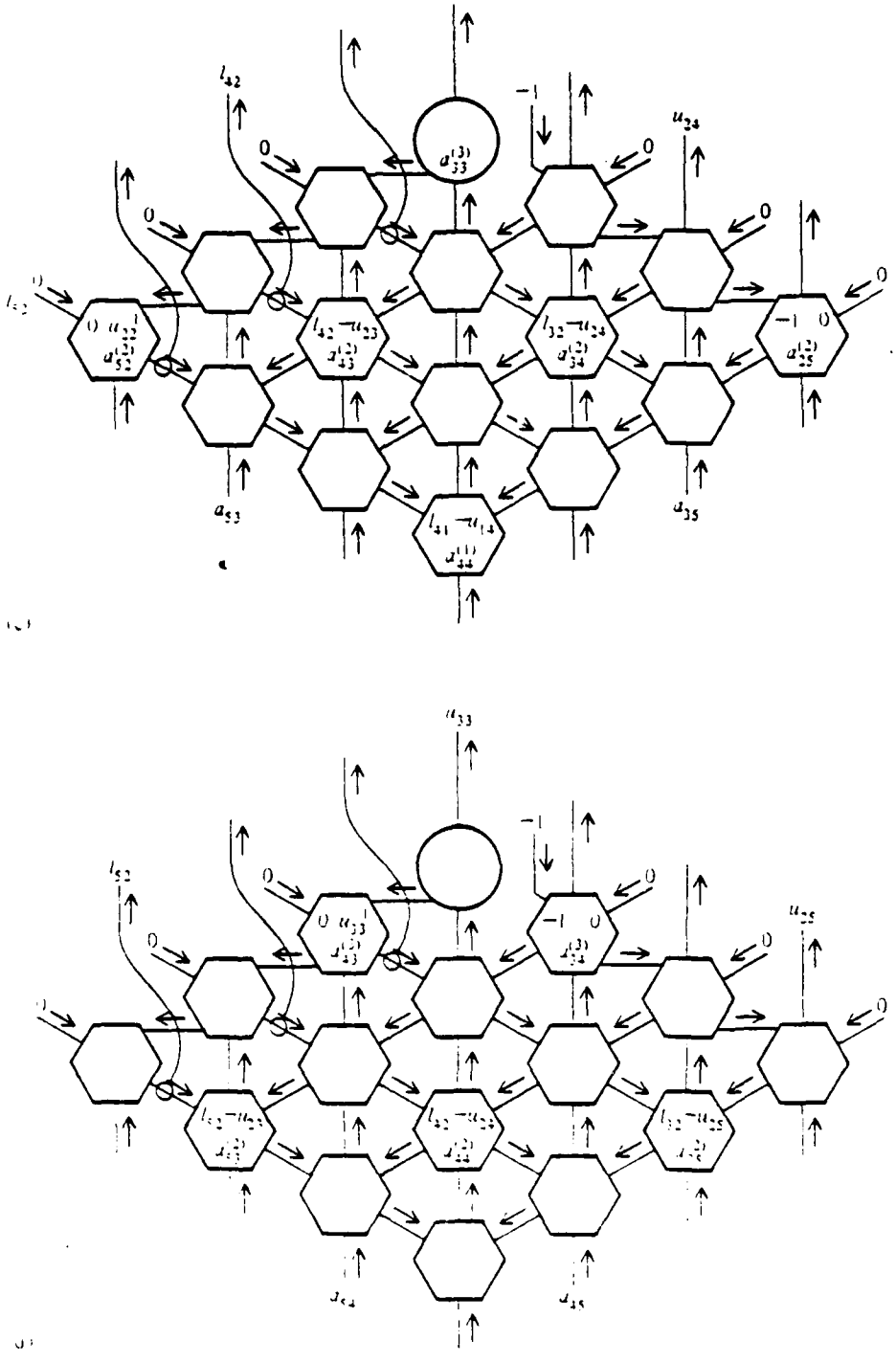


Fig. 17. (cont.)

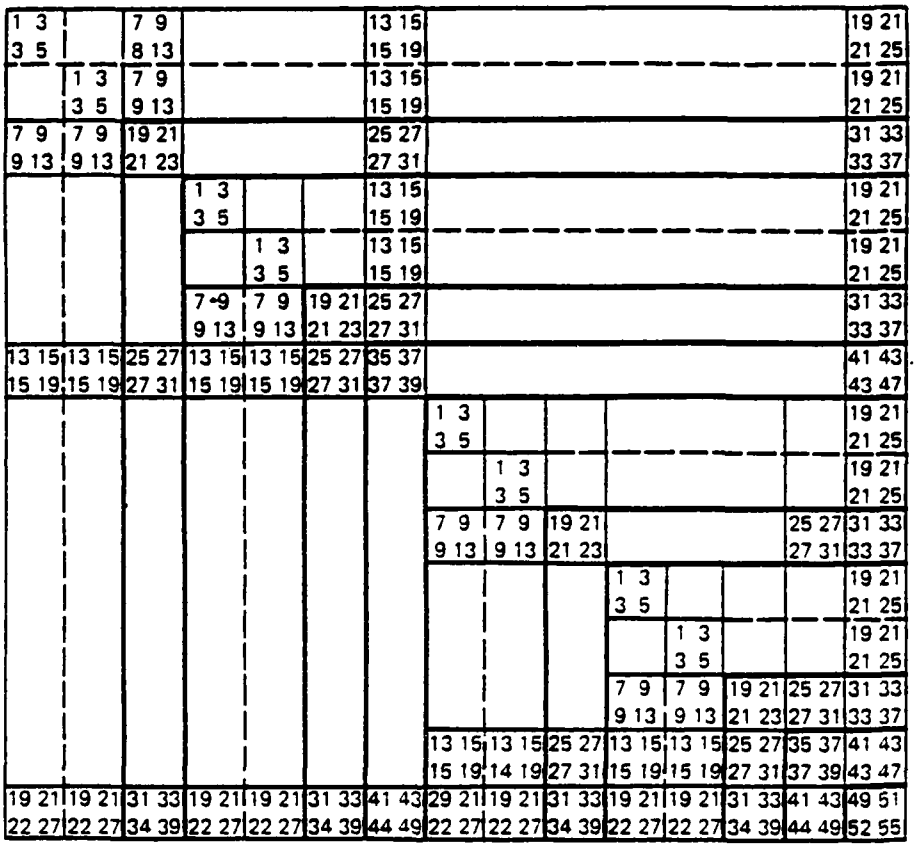


Fig. 18. Input timing for LU decomposition using systolic arrays.

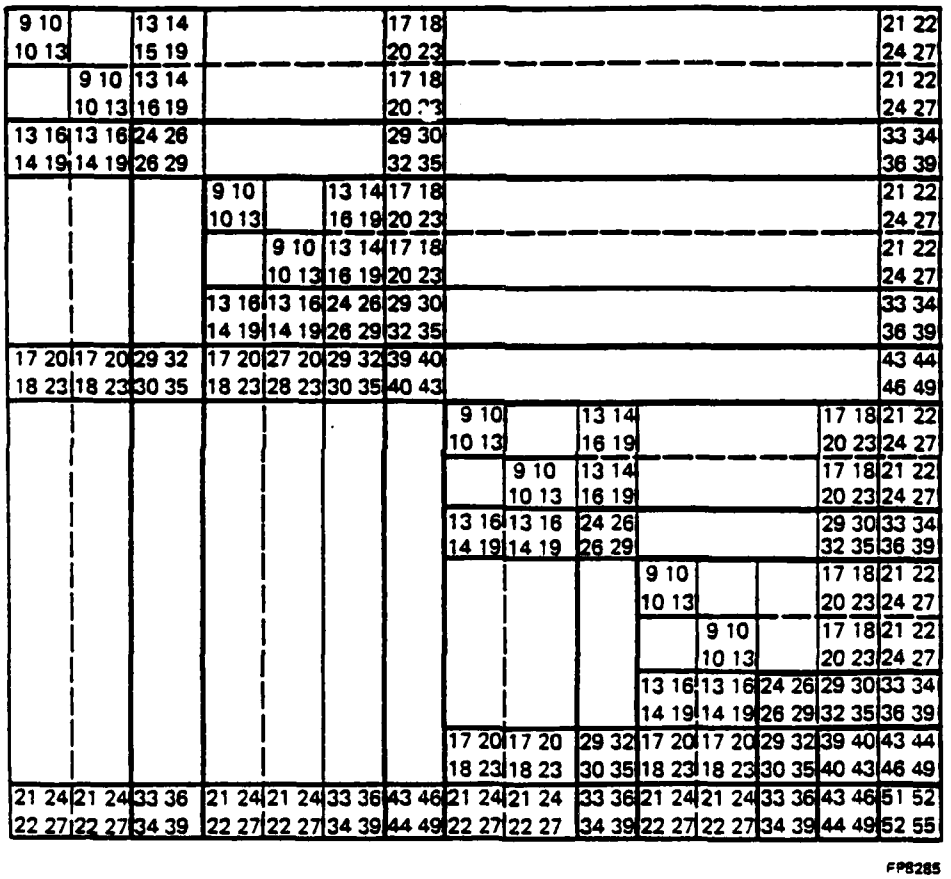


Fig. 19. Output timing for LU decomposition using systolic arrays.

computation, i.e., multiplication and addition, is done until the input data, A, L and U, have their input valid flags on. This distinguishes the normally zero input data elements from the ones at the interconnection location. Also, the partial sums $\sum L*U$ are taken out at different points on the network following a certain pattern. They are then fed into modules which add up all the partial sums from the different parallel subcircuits. The number of interconnection levels depends on the depth of the nested BBDF. Fig. 20 shows the overall configuration of the LU factorization network.

The number of processors used are:

Basic Level: $(m + d*i)^2$

Interconnect 1: $(m + (d-1)*i)^2$

Interconnect 2: $(m + (d-2)*i)^2$

Interconnect d: $(m + i)^2$

i = size of interconnection network (border)

d = depth of the nested BBDF

m = size of the subcircuits

Hwang's scheme for LU factorization is also modified and then applied to the nested BBDF matrix. Both LU decomposition and forward substitution are very easily combined together into the same mesh connected network. The configuration and the input-output timings for a 4 x 4 matrix are shown in Fig. 21.

To solve A = LU and Lz = y, the y vector is put in as an extra column in the A matrix. Then the elements of the matrix are fed into the array, one column at a time. A '1' is fed in at the diagonal

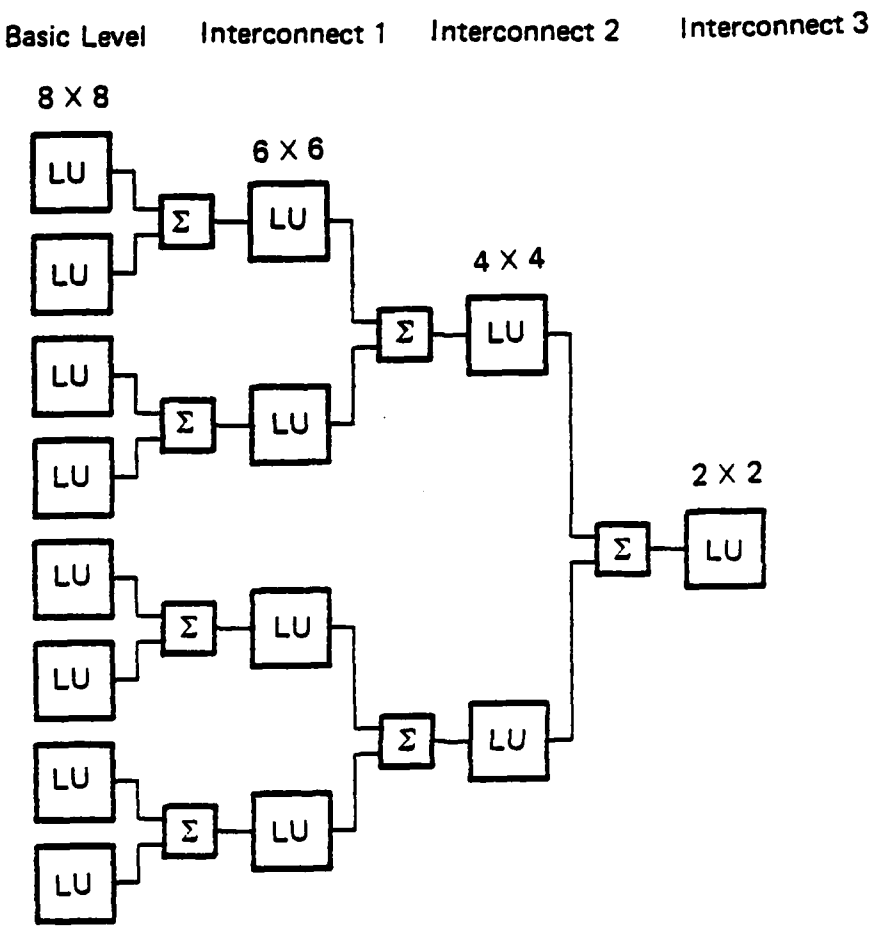
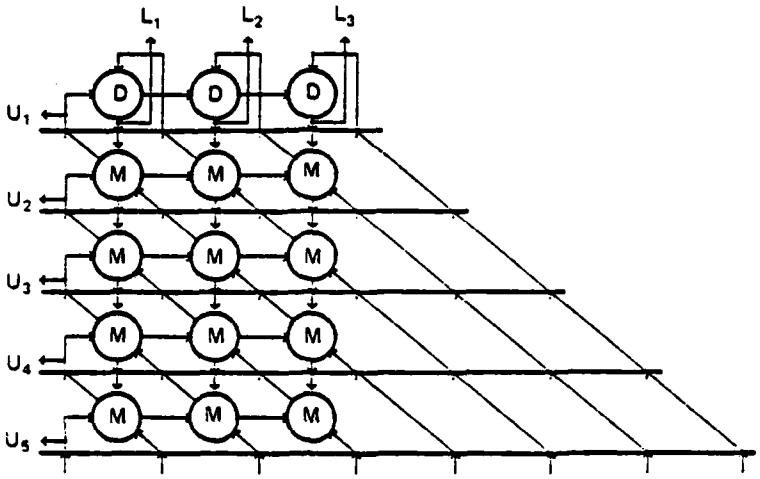


Fig. 20. System configuration using systolic arrays.



Inputs:

t_1	0	0	0	0	a_{11}	a_{21}	a_{31}	a_{41}
t_2	0	0	0	0	1	0	0	0
t_3	0	0	0	a_{12}	a_{22}	a_{32}	a_{42}	0
t_4	0	0	0	0	1	0	0	0
t_5	0	0	a_{13}	a_{23}	a_{33}	a_{43}	0	0
t_6	0	0	0	0	1	0	0	0
t_7	0	a_{14}	a_{244}	a_{34}	a_{44}	0	0	0
t_8	0	0	0	0	1	0	0	0
t_9	y_1	y_2	y_3	y_4	0	0	0	0

Outputs:

U_1	U_2	U_3	U_4	U_5	time	L_1	L_2	L_3
U_{11}	0	0	0	0	t_5	0	0	0
1	U_{12}	0	0	0	t_6	l_{21}	l_{31}	l_{41}
U_{22}	0	U_{13}	0	0	t_7	0	0	0
1	U_{23}	0	U_{14}	0	t_8	l_{32}	l_{42}	0
U_{33}	0	U_{24}	0	z_1	t_9	0	0	0
1	U_{34}	0	z_2	0	t_{10}	l_{43}	0	0
U_{44}	0	z_3	0	0	t_{11}	0	0	0
0	z_4	0	0	0	t_{12}	0	0	0

A = LU and Lz = y

Fig. 21. LU decomposition and forward substitution using the modified Hwang method on a 4 x 4 full matrix.

input position every other clock pulse so that a delay can be put in between two input columns without the elements being changed in the network. The L factors then come out on the upper edge and the U factors from the left edge. Asynchronous handshaking is also applied here, except that the partial sums are taken out right before they enter the division cells on the top row and along the U output latches on the left edge. The partial sums of all the subcircuits within each level are then added to the matrix input elements. The results are the inputs to the next level. The dark lines in between the cells are interface latches to hold inputs and outputs until the next clock pulse arrives. Note also that the system clock must be long enough to accommodate for the time it takes to do the computation or the time it takes to shift the input data to the output. The input and output timings for a 30 x 30 nested BBDF matrix are shown in Figs. 22 and 23, respectively.

For each mesh connected array for LU decompostion (Kai Hwang[10]):

Number of required arithmetic cells = $n^2 - n$

Number of I/O terminal = $4n - 2$

Startup time delay(time delay to get the first output) = $n - 1$

Array net compute time = $2n - 1$

Total compute time = $3n - 2$

If forward substitution is included in the same processor array, together with LU decomposition, the following modifications occur. For each mesh connected array:

Number of required arithmetic cells = $(n+1)(n-1)$

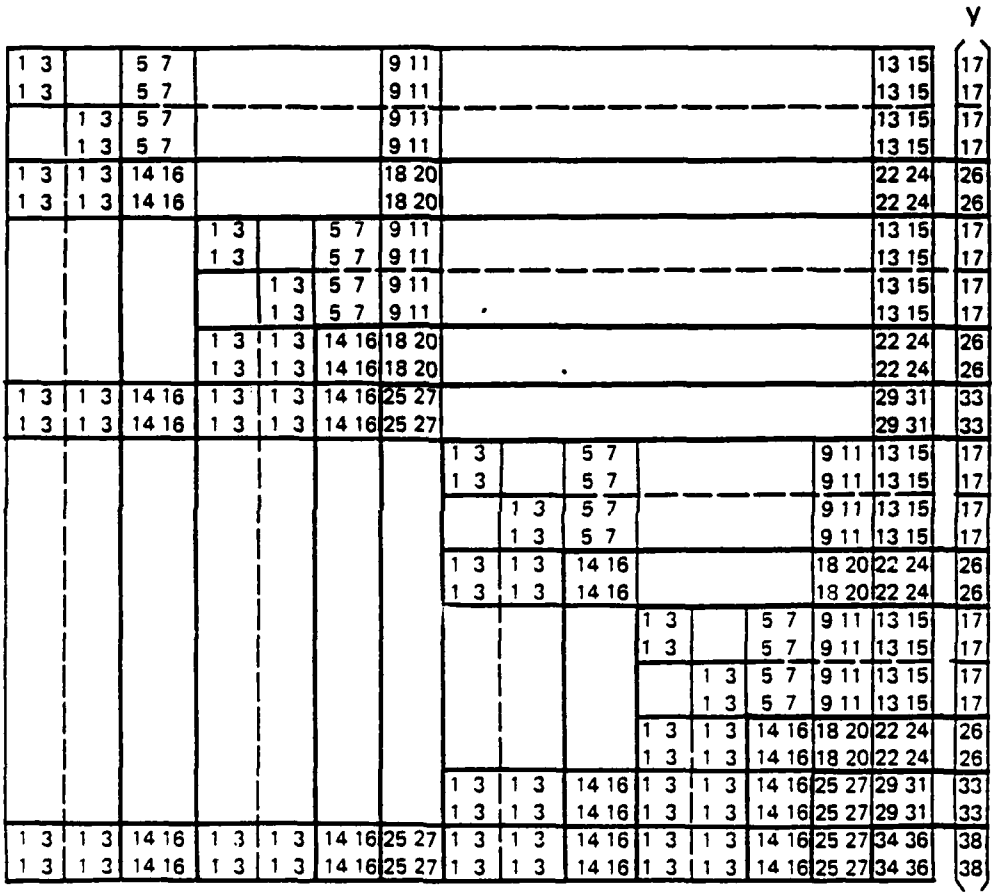


Fig. 22. Input timing for LU decomposition and forward substitution using the modified Hwang method.

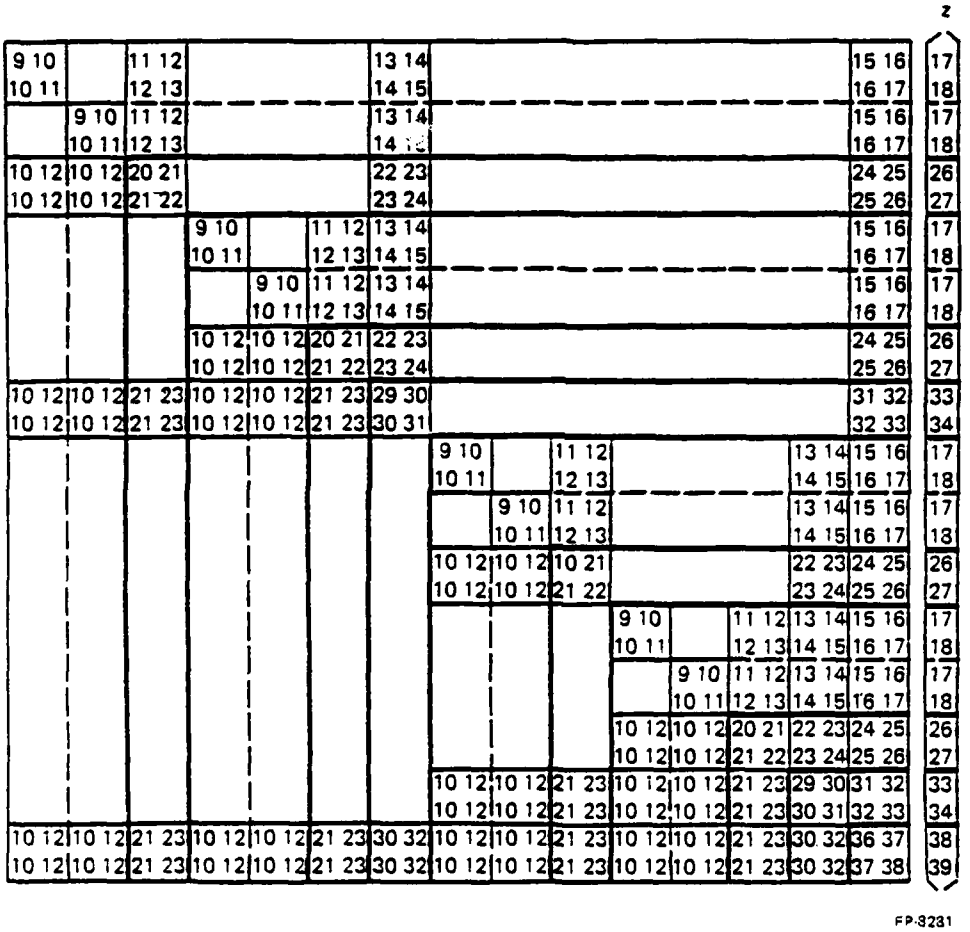


Fig. 23. Output timing for LU decomposition and forward substitution using the modified Hwang method.

Number of I/O terminal = 4n - 1

Startup time delay = n

array net compute time = 2n

Total compute time = 3n

The system configuration is shown in Fig. 24 and the interconnection for a 7 x 5 array to a 5 x 3 array is shown in Fig. 25. Backward substitution uses a linear systolic array (Fig. 26). Solutions are obtained from the bottom of the matrix up to the top. Also, the first computation can only be done after the solutions of the forward substitution are obtained. The input and output timings are given in Fig. 27.

From the results, it can be concluded that the input and output timings of this modified systolic scheme are more regular and easier to control than the hexagonal systolic array. Also, fewer processors and less time are needed to accomplish both LU decomposition and forward substitution. Thus, in the following section the modified Hwang implementation is studied for its performance time.

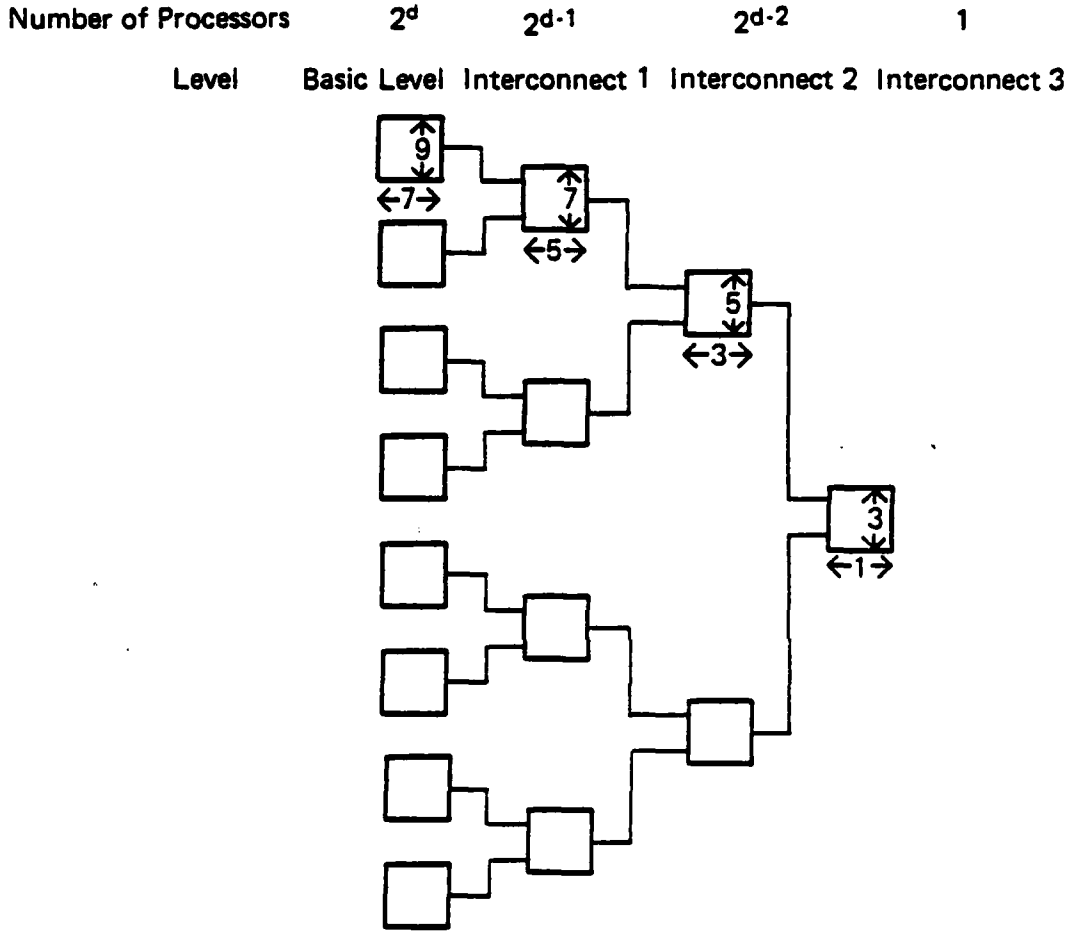


Fig. 24. Processor array configuration for the example nested matrix.

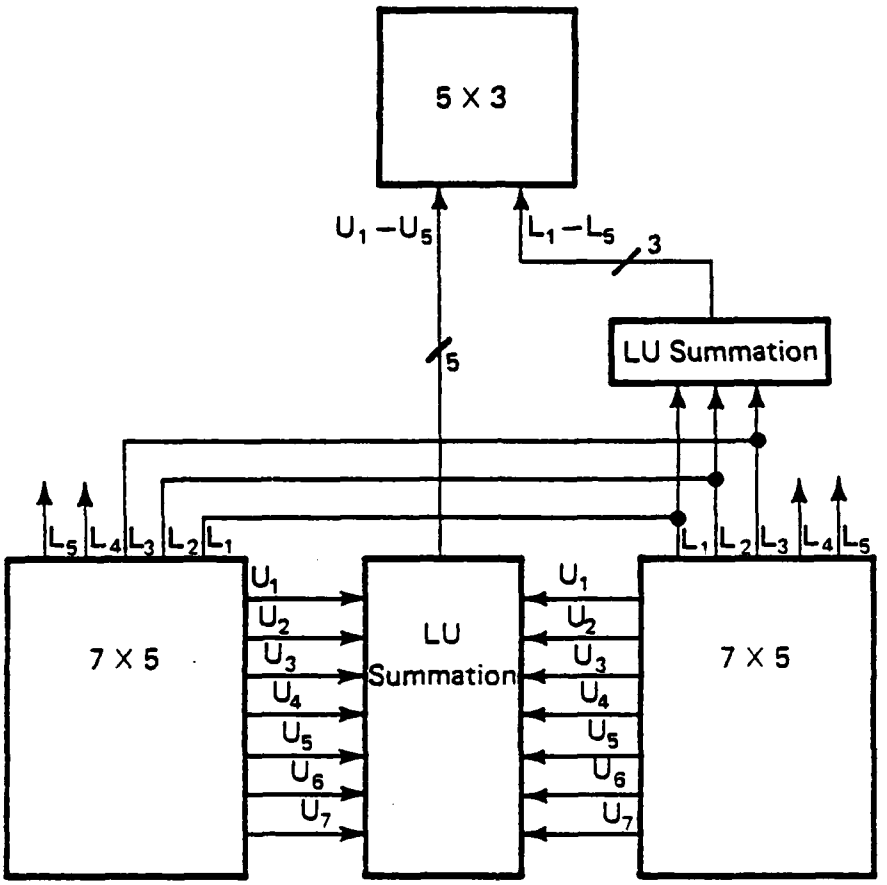
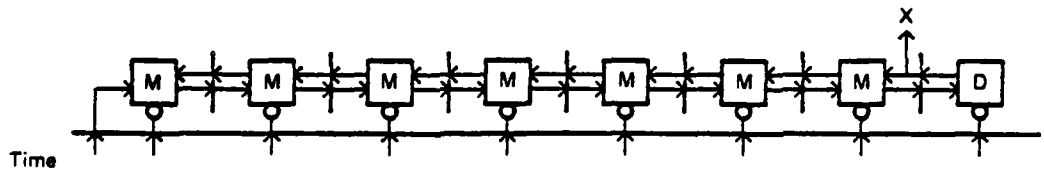


Fig. 25. Interconection between two different levels of processor arrays.



Time									
t_{39}	z_{30}	0	0	0	0	0	0	0	0
t_{40}	0	0	0	0	0	0	0	0	0
t_{41}	z_{29}	0	0	0	0	0	0	0	0
t_{42}	0	0	0	0	0	0	0	0	0
t_{43}	z_{28}	0	0	0	0	0	0	0	0
t_{44}	0	0	0	0	0	0	0	0	0
t_{45}	z_{27}	0	0	0	0	0	0	0	0
t_{46}	z0	0	0	0	0	0	0	0	$U_{30,30}$
t_{47}	z_{26}	0	0	0	0	0	0	$U_{29,30}$	0
t_{48}	0	0	0	0	0	0	$U_{28,30}$	0	$U_{29,29}$
t_{49}	z_{25}	0	0	0	0	$U_{27,30}$	0	$U_{28,29}$	0
t_{50}	0	0	0	0	$U_{26,30}$	0	$U_{27,29}$	0	$U_{28,28}$
t_{51}	z_{23}	0	0	$U_{25,30}$	0	$U_{26,29}$	0	$U_{27,28}$	0
t_{52}	0	0	$U_{24,30}$	0	$U_{25,29}$	0	$U_{26,28}$	0	$U_{27,27}$
t_{53}	0	$U_{23,30}$	0	$U_{24,29}$	0	$U_{25,28}$	0	$U_{26,27}$	0
t_{54}	0	0	$U_{23,29}$	0	$U_{24,28}$	0	$U_{25,27}$	0	$U_{26,26}$
t_{55}	0	0	0	$U_{23,28}$	0	$U_{24,27}$	0	$U_{25,26}$	0
t_{56}	0	0	0	0	$U_{23,27}$	0	$U_{24,26}$	0	$U_{25,25}$
t_{57}	0	0	0	0	0	$U_{23,26}$	0	$U_{24,25}$	0
t_{58}	0	0	0	0	0	0	$U_{23,25}$	0	$U_{24,24}$
t_{59}	0	0	0	0	0	0	0	$U_{23,24}$	0
t_{60}	0	0	0	0	0	0	0	0	$U_{23,23}$

Fig. 26. Linear processor array for backward substitution (Ux=z).

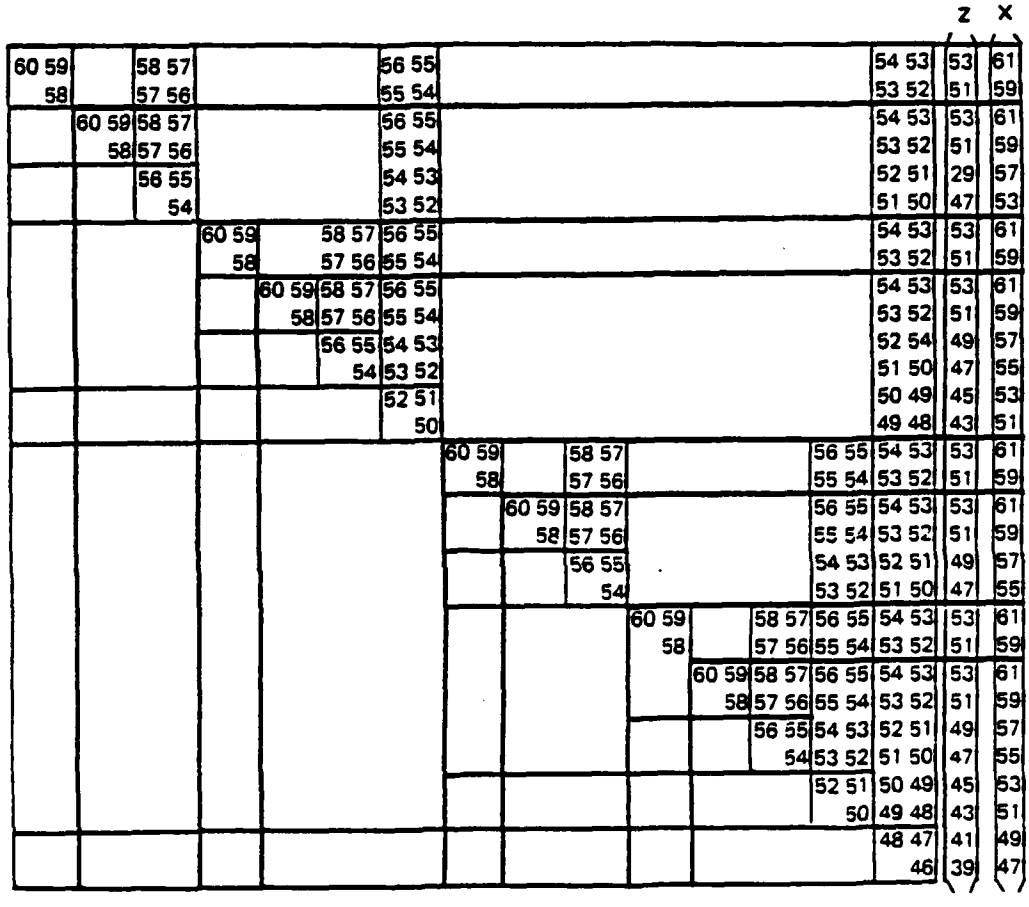


Fig. 27. Backward substitution pipeline (Ux=z) input and output timings.

3.4. Performance Evaluation

Timing Analysis of the Nested Algorithm using the modified Hwang implementation for the example nested matrix:

	Time		
	Start Input	Get LU factors	Get first z
Basic Level	t1	dxi+p = p1	t1+2p1
Interconnect 1	3p1-2(dxi)+2 = t(int1)	t(int1)+(d-1)xi+i (d-1)xi+i = p2	t(int1)+2p2
Interconnect 2	t(int1)+3p2-2(d-1)i+2 = t(int2)	t(int2)+(d-2)xi+i (d-2)xi+i = p3	t(int2)+2p3-1
Interconnect 3	t(int2)+3p3-2(d-2)i+2 = t(int3)	t(int3)+i	t(int3)+2i-1

$T(LU+FWD) = 3\ (d*i + p) + 5i + 6$

$n = 2^d \times i + dxi$

Backward substitution $= 3 * 2^d - 1$

Total time $= 3*(d*i + p) + 5i + 5 + 3*2^d$

Assume processors are very cheap so that the computation time is the main consideration of performance. Note that this network can compute LU factorization in O(n) time. It is thus a very efficient processing array for LU factorization.

CHAPTER 4

DIRECT METHOD - OVERALL SYSTEM CONFIGURATION

There are two main methods for solving large systems of linear equations, the direct and the indirect method. The direct method, as the name implies, feeds the entire matrix into the processor array in one pass, and solves the system of linear equations. The indirect method will be discussed in the next chapter.

The system for the direct method of LU factorization consists of the host interface, the main memory(MM), a control unit(CU), the memory management unit(MMU), the sequencer unit(SU), the feedback buffer(FB), the input processor switch (IPS) and the processor array(PA). The block diagram of the system is shown in Fig. 28.

The host interface communicates with the host computer to get the commands and data to process the LU factorization operation. The matrices are first reordered by reordering programs implementing the nested clustering algorithm in the host computer. The matrices are then represented in partitioned bordered block diagonal form. The non-zero values are then stored with their row and column coordinates. Since the amount of data is large compared to the operation set, the control signals are embedded into the data stream. Also, the host interface recognizes only partitioned matrices and vectors, so the host provides a data separator for each submatrix and vector segment in the data stream. The host interface then generates a

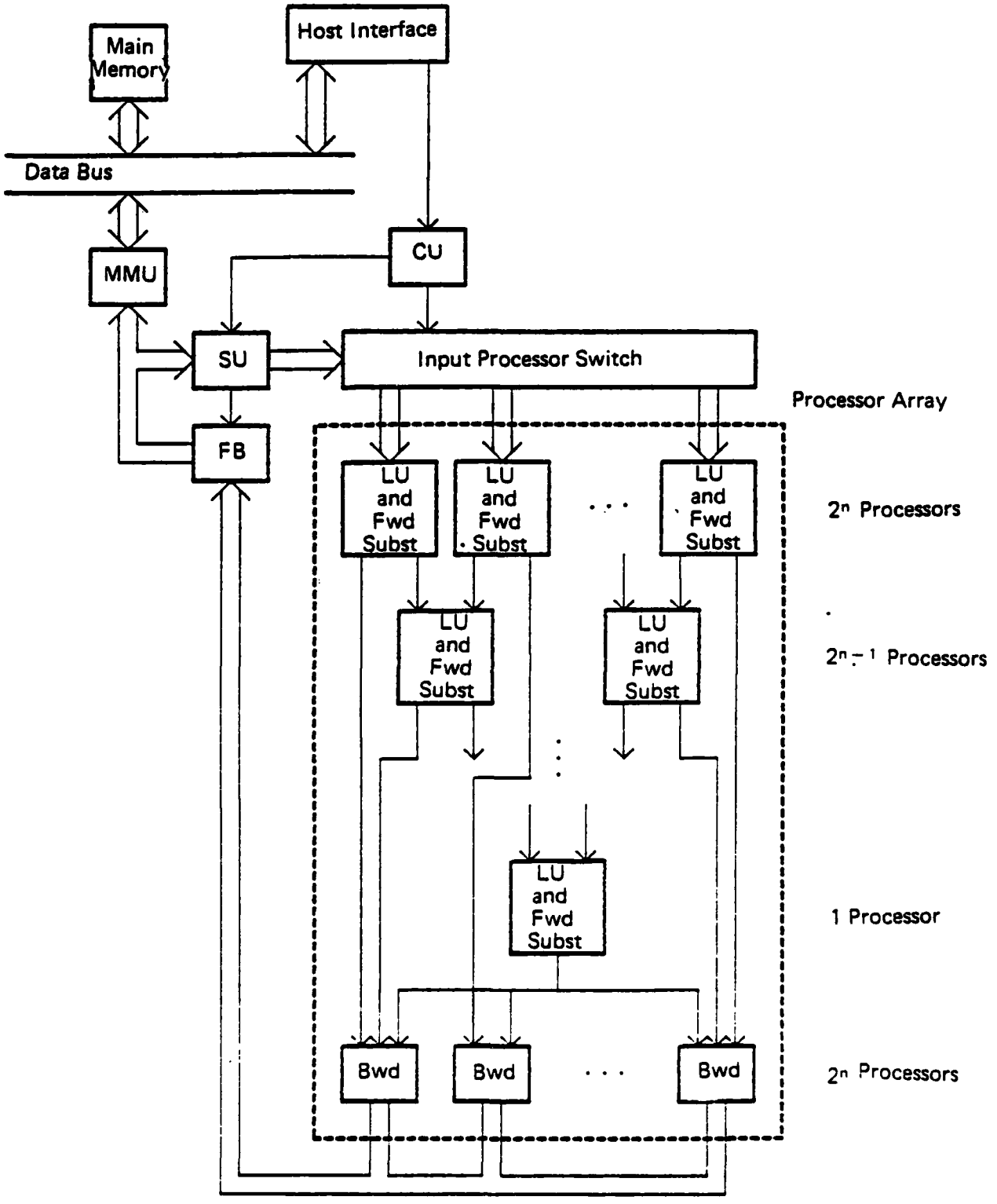


Fig. 28. Basic system configuration for the direct method.

proper data representation each time a separator is encountered. This data representation and the instruction words are then sent to the CU while data is sent to the main memory. The host interface tries to allocate potentially parallel data objects into different parts of the memory hierarchy to be fetched through different ports.

The control unit (CU) decodes the host interface instructions on matrices into sequencer instructions on submatrices and vector segments. Thus each task requested by the host is partitioned into several subtasks which may be carried out in parallel. Each subtask is carried out by the processor array under supervision of the sequencer. The CU thus contains information on the number of nested levels of parallel submatrices as well as the sizes of the submatrices. The control unit also sends commands to the input processor switch to enable and disable processors depending on the size of the matrix to be processed. It also controls the feedback buffer which organizes the output solutions from the processor array.

The memory management unit (MMU) manages the main memory system. A virtual addressing scheme is used because the amount of data of each operand matrix can be very large. The operand address received from the CU is the starting virtual address of a submatrix or vector segment. The MMU computes the ending virtual address from it and translates both virtual addresses to physical addresses. The MMU is required to fetch data for concurrent tasks from the multiport memory. The host interface would try to allocate potentially parallel data objects into different parts of the memory hierarchy to be

fetched through different ports.

The sequencer unit (SU) adds different delays to data loaded by the MMU before sending them to different processor rows or columns depending on the size of the submatrices. The delays are synchronized by a system clock which controls the processor array.

The input processor switch (IPS) receives instructions from the control unit to determine which processor elements are enabled according to the size of the matrices. Then the input data are then automatically switched from the SU to the appropriate processing element.

The feedback buffer (FB) is used when the data output from the processor array is re-sent to the processor array. It also arranges the outputs in an organized fashion to be sent to the MMU for virtual address translation.

The processor array (PA) has been discussed in the previous chapter. The array is controlled by instructions from the CU as well. The processor array shown is the semi-synchronous implementation for the nested BBDF matrices. The processor array is comprised of planes of processing elements; each subsequent plane takes care of the interconnection between two previously parallel arrays for LU factorization and forward substitution. The results are then fed into processing elements used for backward substitution.

This chapter gives the general description of each of the blocks in the special purpose machines used for LU factorization in the

direct method. The details of these blocks still need to be designed.

CHAPTER 5

INDIRECT METHOD

An alternative method for solving large systems of linear equations is an indirect method, such as the Gauss-Seidel [16] relaxation method. Special purpose hardware can also be built to accomplish this algorithm instead of having it done in software.

The block diagram of such a system is shown in Fig. 29. The indirect method first makes a guess on the values of the interconnection nodes. The partitioned submatrices from 1 to k can then be evaluated using LU decomposition, forward and backward substitution. The nested clustering algorithm for partitioning matrices is no longer needed. The matrix can be partitioned in normal BBDF form. The following equations are solved in parallel:

$$(L_1 * U_1)\ x_1 = y_1 - P_1 * x_t$$

.

.

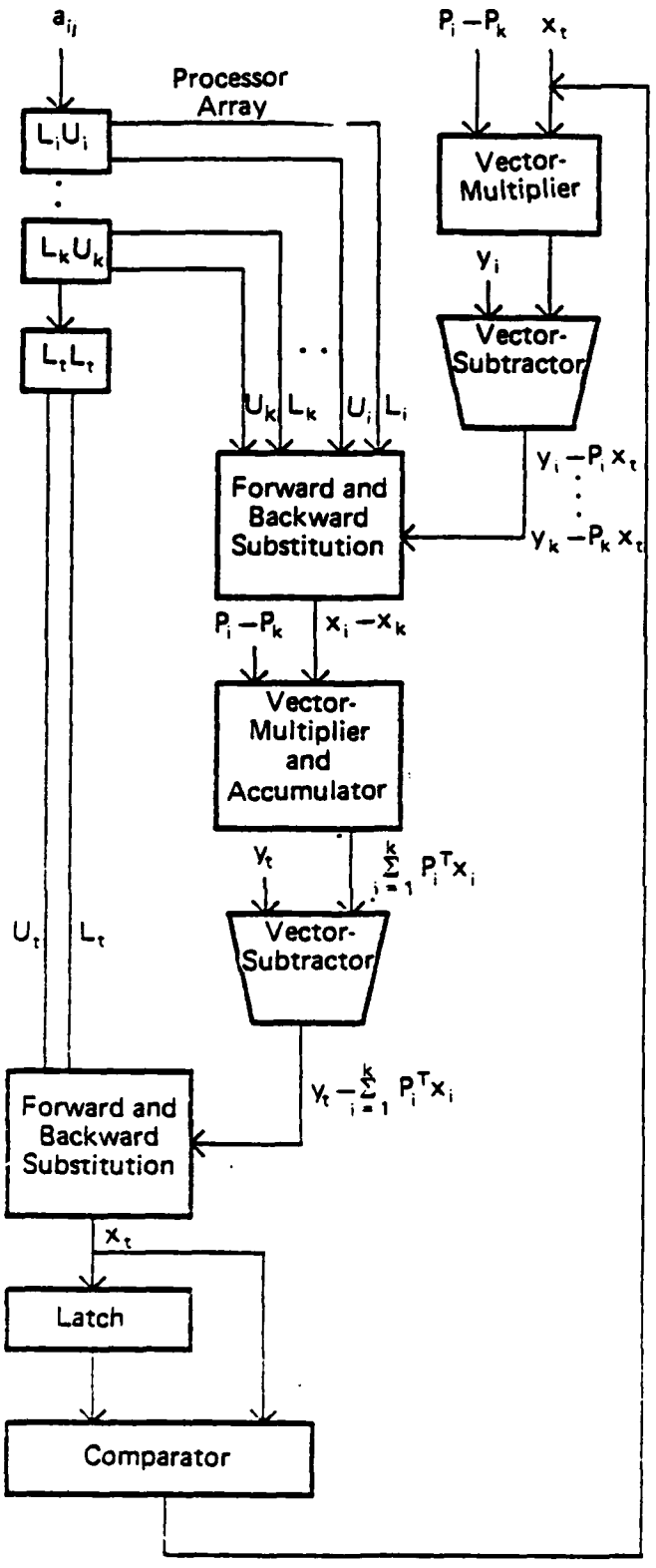
.

$$(L_k * U_k)\ x_k = y_k - P_k * x_t$$

where x_t is the initial guess. Then, the solutions x_1 to x_k are used to compute the values of the interconnection node.

$$(L_t * U_t) x_t = yt - \sum_{i=1}^{k} Pi^T\ x_i$$

After the new values of the interconnection node are computed, they are compared to the previous guess. If the solutions are off by more



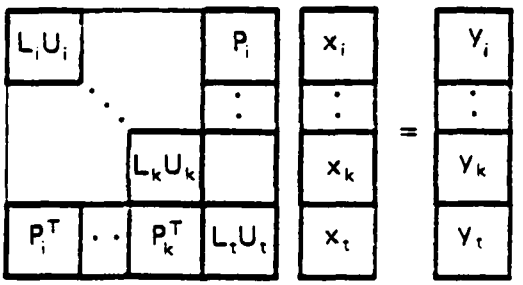


Fig. 29. Block diagram for the indirect method hardware.

than the tolerance allowed, then the new x_t's are used to re-evaluate the values of x_1 to x_k until the solutions converge. Latency can be explored if some of the values of x_1 to x_k are within the tolerance allowed.

The difference between the indirect and the direct method is that the submatrices are completely decoupled from each other in the indirect method because an initial guess for the values of x_t, the interconnection node values, is made. Since each submatrix is connected to another solely by the interconnection nodes, decoupling as a result of guesses, solves the problem of the complexity of controlling the interconnection. However, vector multipliers and subtractors must be added to compute the solutions . A processing element for LU decomposition can be used to find $y_i - P_i * x_t$, whereas a forward substitution array made of the same processing elements can be used to obtain $y_t - \sum_{i=1}^{k} P_i^T * x_i$. In addition, a comparator must be used to compare the values of x_t with the previous iteration to check for convergence.

The direct and indirect method have different applications. If the circuit is approximated with a simplified model, e.g., in RELAX and SPLICE, the indirect method is faster because the solutions converge quickly. However, there are inaccuracies in the solutions. In most cases, these inaccuracies are trivial. The indirect method converges quickly for circuits with very few couplings. However, if a detailed model of the circuit or some complex circuits such as analog

circuits or digital circuits with parasitics, are to be analyzed, then the direct method , e.g., the one used in SPICE [18], is faster and more accurate. Using the indirect method for this application may lead to solutions that do not converge. Thus, both the direct and indirect method are used depending on the types of circuits to be solved.

CHAPTER 6

CONCLUSIONS AND FUTURE RESEARCH

This thesis is a preliminary investigation into a computer architecture to be used for solving large systems of partitioned sparse matrices repesenting the connections of electrical circuits. A highly concurrent parallel architecture is proposed. It is expected to be a powerful tool which is aimed at speeding up the LU factorization of these matrices in order to solve the equations.

A special purpose architecture is chosen over a general purpose architecture. There are many advantages and disadvantages of both types of architectures. However, in spite of the inextensibility of the special purpose architecture, the special purpose architecture is simpler to design and is sufficient for our application. This type of architecture can offer a faster speedup because no time is wasted in decoding instructions. Three different types of special purpose array processors have been studied and compared: the systolic array, the Hwang processor array and the wavefront array processors.

In order to achieve maximum concurrency, the matrix is best ordered in a nested bordered block diagonal form. A modified clustering algorithm is presented based on a heuristic method of ordering these sparse matrices. However, using this algorithm, the number of clusters (or submatrices) that result are always in powers of two. The LU factors at the borders of these nested BBDF matrices are

easier to solve than the normal BBDF. This is because the interconnection nodes are also decoupled using this method, thus making the borders of the submatrices a small size. Separate processor arrays can be used to evaluate the LU factors as well as performing the forward substitution of all the submatrices at the lowest level. The results of two different submatrices can then be summed together and become input to the interconnection processor array in the next level until the entire matrix is computed. Backward substitution can then be applied to the resulting matrix.

In order to facilitate VLSI implementation to reduce cost and computation time as well as to simplify control, a highly modular computing structure with local communications is probably the best strategy. All of the three processor arrays mentioned above, namely, the systolic array, the Hwang processor array and the wavefront array processors have these properties. The systolic array uses a synchronous data flow. The processing elements are all identical with local interconnections. The Hwang processor array uses latches at the border to latch inputs. The wavefront array processors use an asynchronous handshaking scheme for local communications. The synchronous scheme creates problems as the clock skew grows due to the increase in the size of the processor array. However, the asynchronous scheme may cause race conditions and data conflicts if the data is not synchronized correctly. The Hwang processor array makes maximum use of processors; however, latches must be used at the border. This destroys part of the simplicity and regularity of the data con-

trol. To compromise between all the tradeoffs of these architectures, a modified scheme is designed to incorporate the characteristics of all these types of processor arrays. The processor array proposed uses a synchronous data flow to input and output data at each submatrix. A system clock is used to synchronize the data. However, certain handshaking signals (input and output valid flags) are also used so that the computations will not be done unless these flags are present. This scheme essentially adds in wait states and is useful when the outputs flow from one level to the next interconnection level. Several problems , however, may arise. Race conditions and conflicts may occur. The data must be able to be latched in each processing element long enough before all the valid signals are present. Also, the input data must be monitored so that data will not be overwritten while waiting for the valid signals.

The Hwang VLSI structures are also included in the processor array because latches are used to buffer the data instead of using registers inside the processing elements. This implementation offers maximum processor utilization. In addition, the inputs are fed in column by column and are thus easily controlled. The processor array proposed can solve LU decomposition and forward substitution in O(n) time. Backward substitution is implemented by a linear array which can also compute in O(n) time.

There are two methods for solving a large system of linear equations, the direct method and the indirect method. The direct method solves the entire matrix in one pass. The indirect method

guesses at the values of the interconnection nodes initially to solve for the rest of the circuit. Then the values of the interconnection nodes are updated and the whole process is repeated until the solutions converge. It is still unclear which method is actually better in terms of speed. The performance of the indirect method depends on how good the initial guess is so that the solutions converge quickly provided they converge. A general description of the overall system configuration of both methods has been given. This thesis deals with the processor array in detail. The design of a single processor element is also given in the Appendix. However, the hardware and the software controls of the rest of the system have yet to be designed in detail.

A simulator will be written to simulate the data flow between all submatrices and their interconnection levels. The results from this simulator give the time when data is input and output at all the processing elements. These results are useful for monitoring the control of the data for any size of matrix. A hardware descriptive language [17], which implements the basic instruction set needed to control the processing elements, must be defined. The language can be similar to the wavefront-oriented language by S.Y.Kung [13], but it should be more special purpose and simple to design. Another simulator can then be written to describe the subroutines used to synchronize the processing elements. This simulator describes the sequence of instructions that each processing element recieves. Data can be fetched from and flowed to any adjacent cells in the up, down,

left or right directions. Conflicts and deadlocks can be avoided [17] if every occurrence of a FLOW ⟨to direction⟩ instruction of a data sourcing PE is matched by an occurrence of a FETCH ⟨from opposite direction⟩ instruction in the instruction sequence of the appropriate recieving PE, i.e., every FLOW is matched both in number of occurrences and sequencing of appearance to a FETCH in the same phase. Data bus contention problems must also be solved. The nested clustering algorithm can also be implemented to reorder the input matrices. Uneven depths of clusters can be incorporated as well.

In terms of hardware, the processor arrays can be designed so that they can deal with matrices of any arbitrary size, either larger or smaller than the hardware can accommodate. Larger matrices need to be partitioned into smaller matrices while smaller matrices are patched with 1's in the diagonal and 0's in the rest of the rows and columns. Once the software is finished, the details of all the other blocks in the system configuration can be implemented. A floating point processing element can be designed in CMOS so that the array can be used for circuit simulation. The CMOS cell will contain 24 bits for the mantissa and 8 bits for the exponent. Local memory, e.g., a RAM, can be added into each processing element. This reduces the memory access time and can thus yield higher performance.

Fast simulation engines can be the key to improving circuit simulation time once a good algorithm has been developed. LU factorization is one of the time-consuming loops in simulation programs.

The design of highly concurrent parallel architectures for this application can undoubtedly revolutionize circuit simulation.

APPENDIX: DESCRIPTION OF A PROCESSING ELEMENT

A 16-bit fixed point processing element (PE) has been designed using a 3um CMOS VLSI technology. The PE accepts 16-bit operands, with 8 bits representing the fractional part and 8 bits representing the integer part. The four major functions that it performs are:

$$(1)\ A_{ij}^{m+1} = A_{ij}^{m} - L_{ik}^{m} U_{kj}^{m}$$

$$(2)\ L_{ik}^{m} = A_{ik}^{m} / U_{kk}^{m}$$

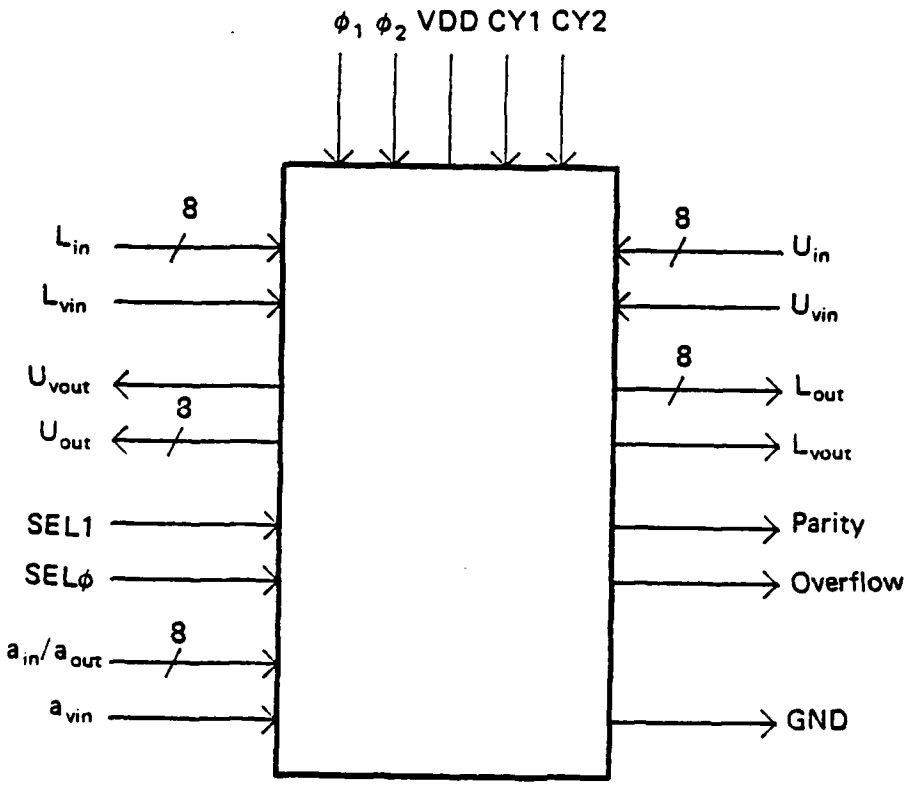
$$(3)\ U_{kj}^{m} = A_{ij}^{m}$$

$$(4)\ A_{ij}^{m+1} = A_{ij}^{m} + L_{ik}^{m} + U_{kj}^{m}$$

Function (1) performs the update of the matrix elements during LU factorization. Function (2) solves for the L (lower triangular) factors while function (3) solves for the U (upper triangular) factors. Function (4) is used when addition is needed at the interconnection level.

The cell basically consists of a fixed-point adder, a multiplier, a divider, a function decode, several muxes and latches. The pin specifications of the chip are given in Fig. 30. The functional block diagram of the cell is shown in Fig. 31. The cell is designed in CMOS because of low power dissipation which is important in an array of large numbers of processing elements.

The design of the cell is semi-synchronous, level sensitive and combinational. It is semi-synchronous because Avin, Lvin and Uvin are flag signals to enable the computation (i.e., the addition and



Total Number of Pins = 55 pins

Pin Assignment:

a_{in} – $a_{ij}^{(k)}$
– Input/Output Pin (8 bits)

a_{vin} – $a_{ij}^{(k)}$ Valid
– Input Pin

U_{in} – $U_{kj}^{(k)}$
– Input (8 bits)

U_{vin} – $U_{kj}^{(k)}$ Valid
– Input

L_{in} – $l_{ik}^{(k)}$
– Input (8 bits)

L_{vin} – $l_{ik}^{(k)}$ Valid
– Input

	SEL1	SELϕ
Aout = Ain – LinUin	0	0
Lout = Ain/Uin	0	1
Uout = Ain	1	0
Aout = Ain + Lin + Uin	1	1

ϕ1(Ph1)
ϕ1(Ph1)
ϕ2(Ph2)
Cycle 1(CY1)
Cycle 2(CY2)
} Clock Inputs

Parity – Output Pin
–Either Adder or Multiplier Parity

Overflow – Output Pin
– Either Adder or Multiplier Overflow

Lout, Uout, Aout – Output Pins for L_{ik} $L_{ik}^{(k+1)}$, U_{π}
Output
Output Pins for $L_{ik}^{(k+1)}$, $U_{kj}^{(k+1)}$, and $A_{ij}^{(k+1)}$

Lvout, Uvout, Avout – High during CY2 and the valid signals (Lvin, Uvin and Avin respectively) are high
– Output Pins

Fig. 30. Pin assignment of a CMOS processing element.

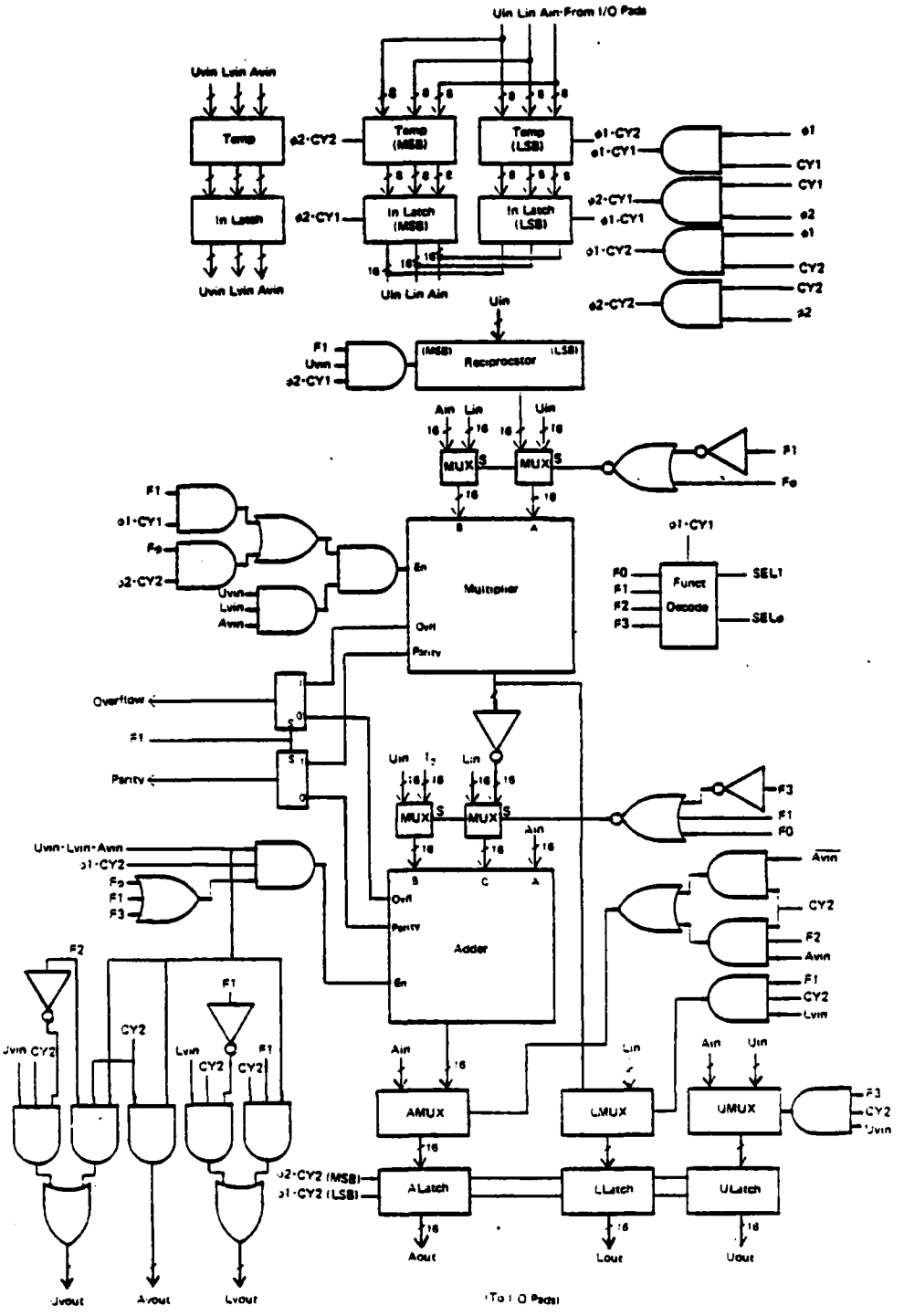


Fig. 31. Logic block diagram for the CMOS processing element.

multiplication) when the system clock goes high. It is level sensitive because it responds to the logic level of the clock instead of the clock edge. Two cycles of a two-phase clock are used for each computation of a processing element. It is combinational because no feedback path exists within the cell so that a finite state machine is not needed.

The inputs of A, L and U are time-multiplexed into the circuit because of the limitations on the number of I/O pins. This scheme, however, slows down the fetching of operands. The entire operation, fetching of data and computation takes two cycles. The timing diagram of the control signals and the data is shown in Fig. 32. During the first cycle (CY1), phase 1 latches the least significant bit LSB data (Ain, Lin, Uin, Avin, Lvin, Uvin) from the temporary latches while the most significant bit MSB data are latched in phase 2. The LSB and MSB temporary latches are loaded in $\phi 1$ and $\phi 2$ of the previous cycle 2, respectively, from the output of an identical adjacent processing element. Also, the input Ain and output Aout share the same I/O pins because of pin limitation and the fact that the next inputs are streamed from another chip at one or more clock cycles after the outputs are generated in this chip. Thus, no I/O conflicts should occur if data are properly synchronized in the processor array. Lin and Uin, on the other hand, cannot share the pins with Lout and Uout because data can be input and output at the same time during an operation.

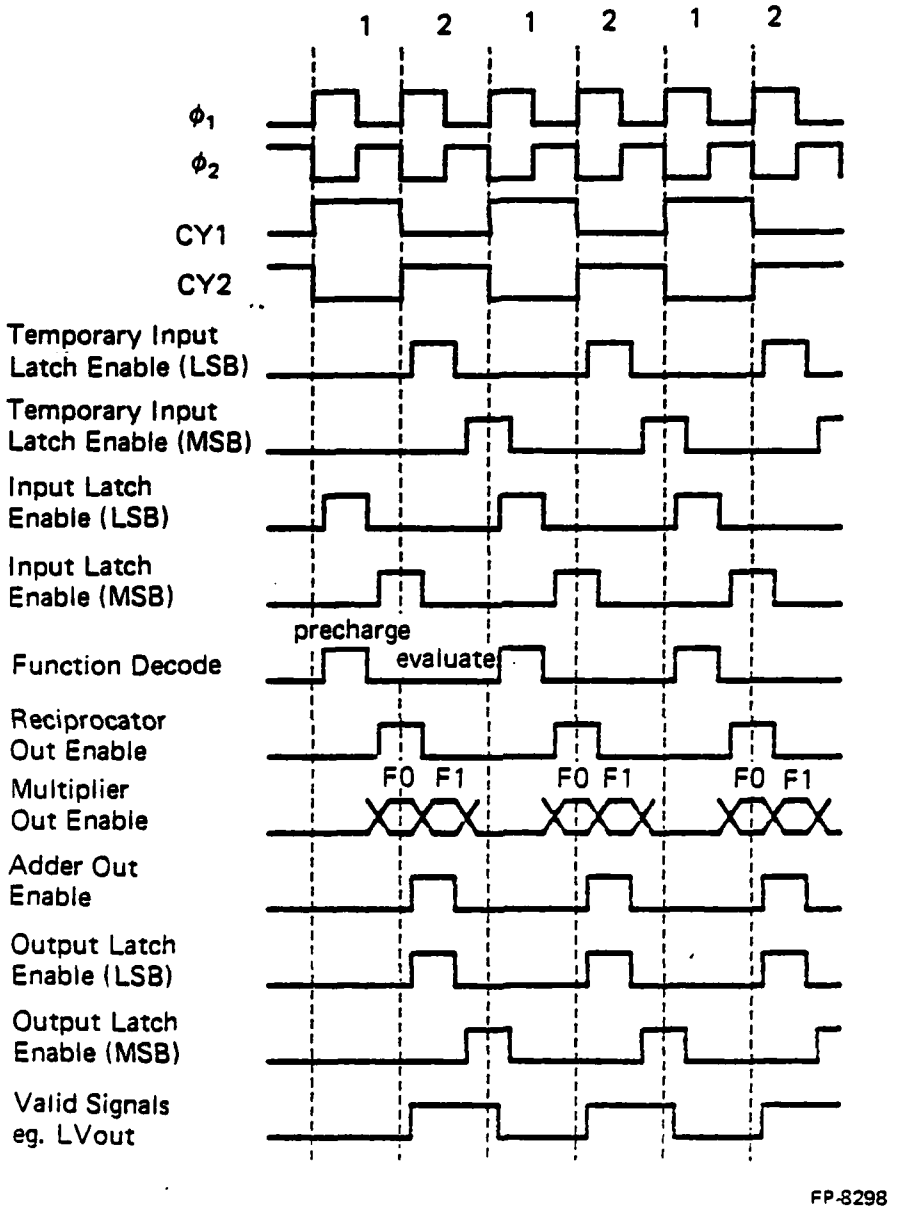


Fig. 32. Timing of the control signals and the data.

A domino CMOS design (Fig. 33) is used for the function decode instead of the conventional dual NAND and NOR gates because it is simpler to design and takes up less area. The function decode is precharged during $\phi 1$ and CY1 and the function is evaluated when $\phi 1$ and CY1 go low, i.e., when $\phi 2$ and CY1 are high. Then SEL1 and SEL0 must be held at the same voltage until the next $\phi 1$ and CY1, otherwise the function selected will be changed during this operation. The computation is somewhat pipelined by two different pipes.

If a multiply-addition (F0) is selected the multiplier outputs are enabled during $\phi 2$ and CY1. As soon as the MSB of the data get latched, the computation will be done right before CY1 goes low. Then the adder is enabled at $\phi 1$ and CY2 and results are latched in the output multiplexer (mux). The LSB of the output is generated at $\phi 1$ and CY2 and the MSB of the output at $\phi 2$ and CY2. If a division is selected (F1), the reciprocator output is enabled during $\phi 2$ and CY1 and the results are processed during $\phi 1$ and CY2. Then the results are carried to the L latches.

No results from the adder, multiplier and reciprocator are latched if the valid signals of all (A, L, and U) have not arrived. After they arrive, the valid signals being passed (as output) to the next cell are enabled during cycle 2.

Parity and overflow are both generated by the multiplier and adder, and when reciprocation is done, the parity and overflow of the multiplier is output whereas those of the adder are normally being generated.

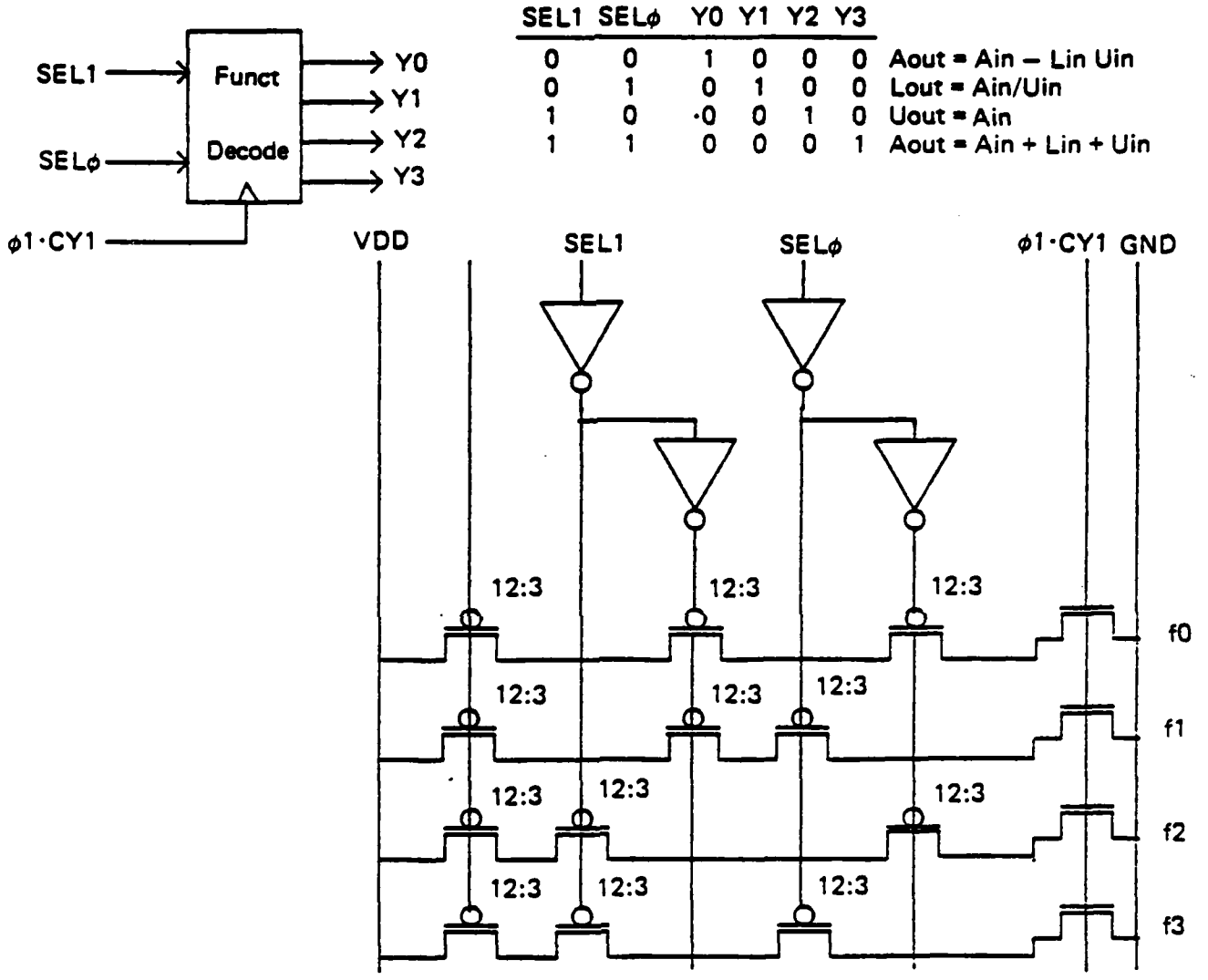


Fig. 33. Function decoder with domino CMOS design.

Two schemes were considered for the design of the adder, a ripple-carry and a carry-look-ahead. Ripple-carry is extremely slow because the carry has to propagate to the previous full adder before the full adder can perform the addition. The adder on the chip has the following functions:

(i) Ain + Lin + Uin

(ii) Ain - Lin * Uin

Thus a three-operands to two-operands adder is put in front of a carry-look-ahead adder. The carry-look-ahead adder is shown in Fig. 34. The parity and overflow are also generated.

The input muxes to the multiplier are enabled so that when the function is F0, Uin * Lin is performed while Uin^{-1} * Ain is performed when the function is F1. The input muxes to the adder are enabled such that when the function is not F3, then Ain - Lin * Uin is performed except for F2 when the output of the adder is not used. During F3, Ain + Lin + Uin is performed and the inputs are chosen as appropriate.

The L output is valid when the function is F1 and Avin, Uvin and Lvin are asserted and the clock cycle is cycle 2; or function is not F1 and Lvin is valid and the clock cycle is cycle 2. The U output is valid when function is F2 and Avin, Uvin and Lvin are asserted and the clock cycle is cycle two; or the function is not F2, but Uvin and cycle 2 are high. The A output is valid when Avin and Lvin and Uvin are high and the clock cycle is cycle 2.

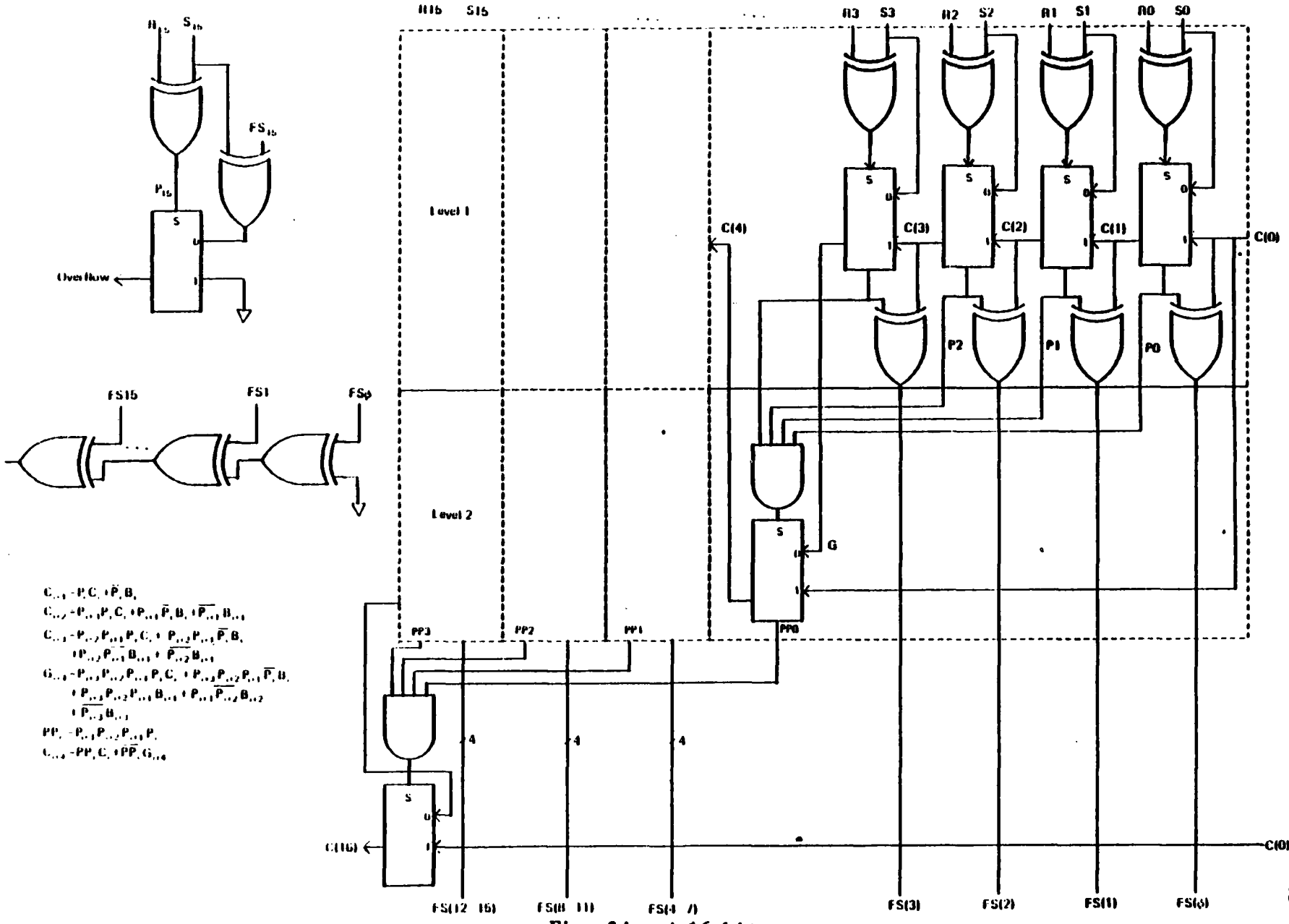


Fig. 34. A 16-bit carry-look-ahead adder.

The reciprocator is based on a non-restoring division principle [23]. The block diagram of the reciprocator and a basic cell in the reciprocator are shown in Figs. 35 and 36, respectively. Each cell performs the functions:

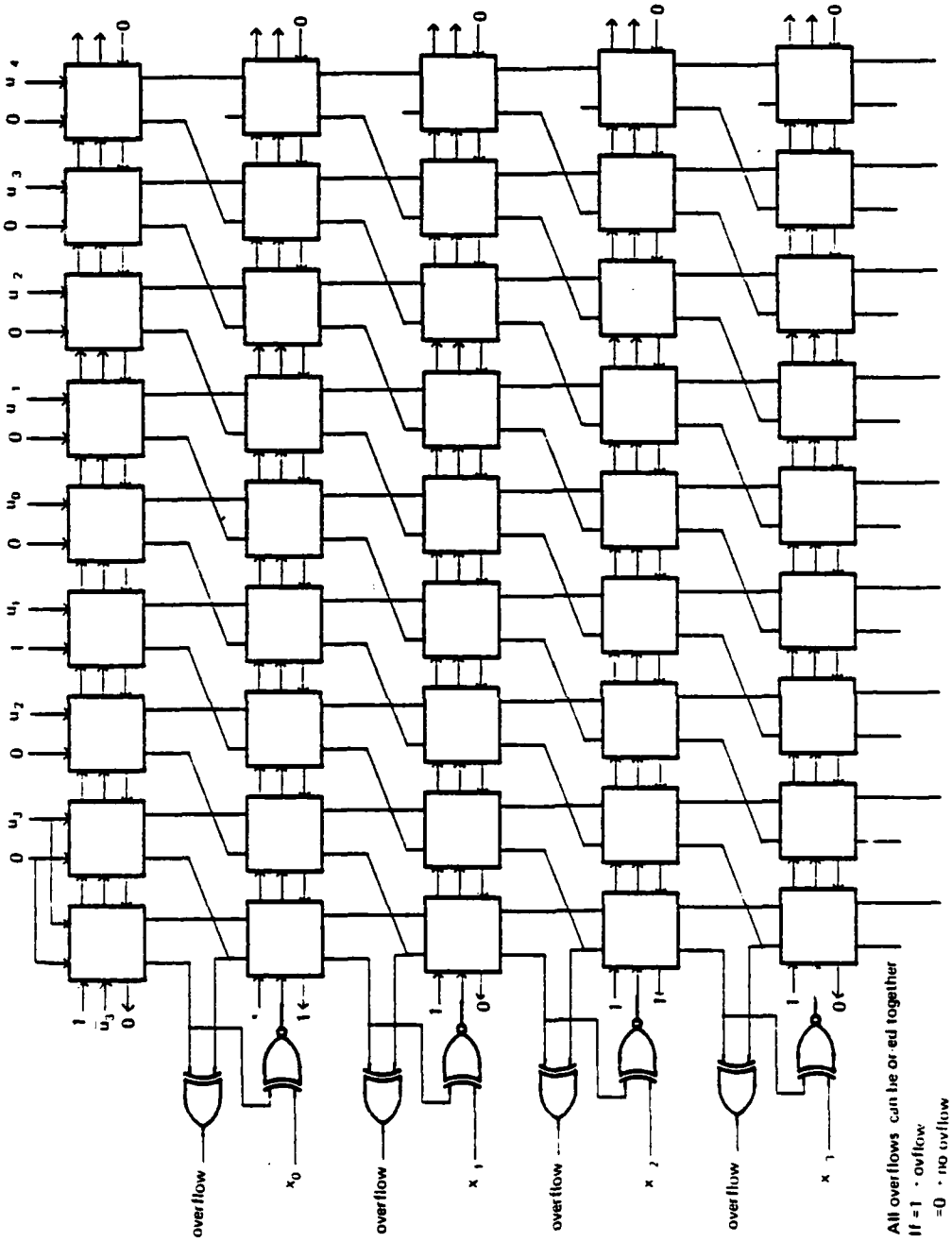
$$z = x \text{ xor } [a \ (\ y \text{ xor } t)]$$

$$u = (x \text{ xor } b) \ (y + t) + yt$$

The multiplier (Fig. 37) is a two's complement combinational array multiplier based on Booth's algorithm [23]. The main cell is basically the same as the reciprocator cell. An extra column of cell is added to the 16 x 16 cells to generate the true sign of the partial products. Also, the last cell of this column can be xor'ed with the MSB of the 16 x 16 multiplier result to generate the overflow check.

A two-phase clock is used with two cycles in each phase. Phase one must be long enough for both the adder to finish addition or for the multiplier to finish multiplication and also for the output latch to latch in the LSB. Phase two must be long enough for both the input latch to latch the MSB or the output latch to latch the MSB and also for the reciprocator to finish calculation.

Note that the multiplier scheme using Booth's algorithm calculates in moderate speed and it takes up a lot of area. A faster multiplier can be implemented. In addition, reciprocation and then multiplication can be replaced by just a divide using a divider so that the calculation needs only go through one stage. Also, $\phi 1$ can be shortened too. A RAM can also be added so that intermediate results



$\frac{1}{0.0010.0000} = 0.100 = -\frac{1}{2} = \frac{1}{2}$

$\frac{1}{1.1100.0} = 1.1110$

Fig. 35. The reciprocator.

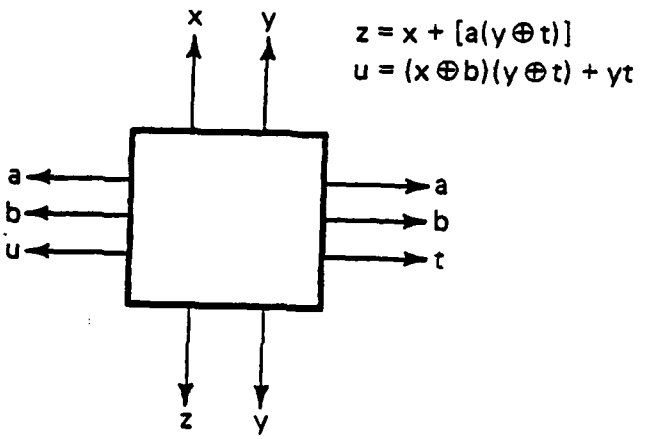


Fig. 36. A basic cell for the reciprocator.

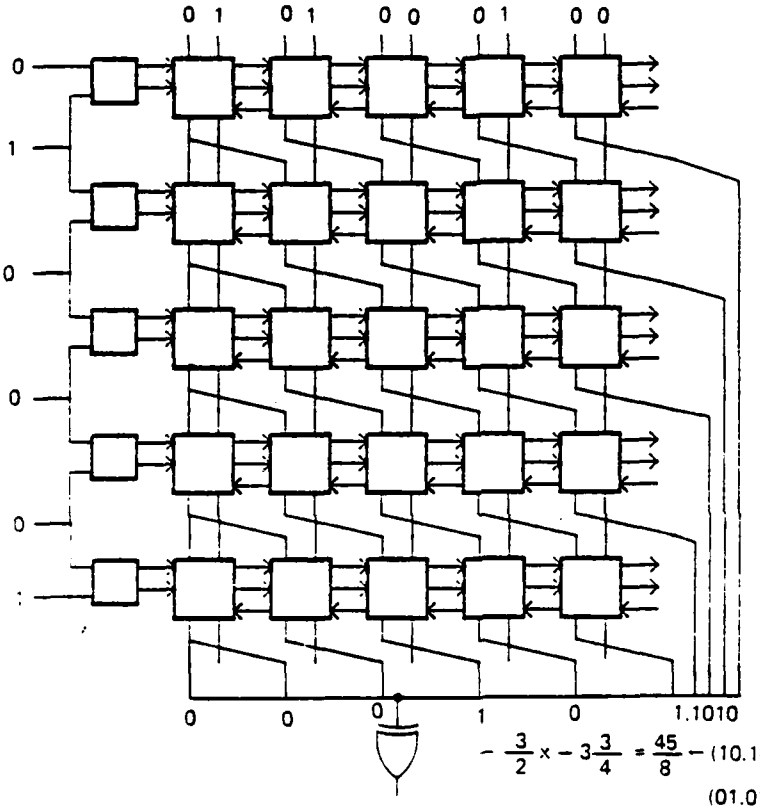
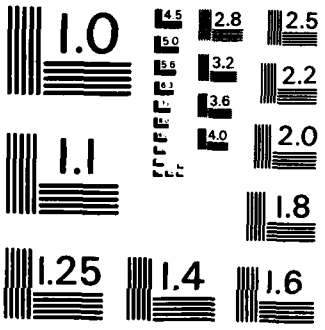


Fig. 37. The multiplier using Booth's algorithm.

AD-A161 351 SPECIAL PURPOSE COMPUTER ARCHITECTURE FOR LU FACTORIZATION OF PARTITIONED SYSTEMS(U) ILLINOIS UNIV AT URBANA COORDINATED SCIENCE LAB K I LUI AUG 85 2/2

UNCLASSIFIED R-1015 N00014-84-C-0149 F/G 9/2 NL



can be stored in local memory and can shorten memory access time. In addition, a ROM can also be added to store the instruction sequences of each processing element. A floating point processing element can be designed using the same functional control described in this appendix.

REFERENCES

[1] F.H.Ko and A.Sangiovanni-Vincentelli, 'BLOSSOM: An algorithm and architecture for the solution of large-scale linear system," IEEE International Conference on Computer Design: VLSI in Computers, pp. 400-403, October 31, 1983.

[2] S.Y.Kung, K.S.Arun, R.J.Gal-Ezer, D.V.Bhaskar Rao, 'Wavefront array processor: language, architecture and applications," IEEE Transactions on Computers, vol.C-31, no.11, pp. 1054-1066, November 1982.

[3] D.W.L.Yen and A.V.Kulkarni, 'The ESL systolic processor for signal and image processing," IEEE Proceedings, pp. 265-272, 1981.

[4] H.T.Kung, 'Why systolic architectures?" IEEE Computer, vol. 15, no.1, pp. 37-48, January 1982.

[5] I.N.Hajj, 'Sparsity considerations in network solution by tearing," IEEE Transactions on Circuits and Systems, vol. CAS-27, no.5, pp. 357-366, May 1980.

[6] K.Hwang and Y.H.Cheng, 'VLSI computing structures for solving large-scale linear system of equations," Proceeding of the International Conference on Parallel Processing, IEEE Catalog no. 80 CH1569-3, pp. 217-230, August 26-29, 1980.

[7] R.M.Kieckhafer and C.Pottle, 'A processor array for factorization of unstructured sparse matrices," IEEE International Conference on Circuits annd Computers, ICCC 82, pp. 380-383, September 28 - October 1, 1982.

[8] M.A.Franklin and D.F.Wann, 'Asynchronous and clocked structures for VLSI based interconnection networks," The Ninth Annual Symposium on Computer Architecture, Austin, Texas, pp. 50-59, April 1982.

[9] M.J.Foster and H.T.Kung, 'The design of special purpose VLSI chips," Computer Magazine, IEEE Computer Society, pp. 26-40, January 1980.

[10] K.Hwang and Y.H.Cheng, 'Partitioned algorithms and VLSI structures for large-scale matrix computations," Proceeding of IEEE Computer Society 5th Symposium on Computer Arithmetic, Ann Arbor, MI, pp. 220-230, May 1981.

[11] D.P.Siewiorek, C.G.Bell and A.Newel, Computer Structures: Principles and Examples. New York: McGraw-Hill Book Co., 1982, chap. 14.

[12] P.M.Kogge, The Architecture of Pipelined Computers. New York: McGraw-Hill Book Co., 1981, chap. 4.4.5.

[13] C.Mead and L.Conway, Introduction to VLSI Systems. Menlo Park, California: Addison-Wesley Pub. Co., 1980, chap. 8.

[14] S.Y.Kung, "On supercomputing with systolic/wavefront array processors," IEEE Proceedings, vol. 72, no. 7, July 1984, to be published.

[15] A.Sangiovanni-Vincentelli, L.K.Chen and L.O.Chua, "An efficient heuristic cluster algorithm for tearing large-scale networks," IEEE Transactions on Circuits and Systems, vol. CAS-24, no. 12, pp. 709-717, December 1977.

[16] A.R.Newton, A.Sangiovanni-Vincentelli, "Relaxation-based electrical simulation," IEEE Transactions on Electron Devices, vol. ED-30 no. 9, pp. 1184-1207, September 1983.

[17] R.J.Gal-Ezer, "The wavefront array processor and its applications," Ph.D. dissertation, Dept. of Electrical Engineering, University of Southern California, December 1982.

[18] W. Nagel, "SPICE2, A computer program to simulate semi-conductor circuits," Memo No. ERL-M520, University of California, Berkeley, May 1975.

[19] A.R.Newton, "Timing, logic and mixed-mode simulation for large MOS integrated circuits," Computer Design Aids for VLSI Circuits, The Netherlands: Sijithoff and Noordhoff, pp. 175-240, 1981.

[20] E. Cohen, "Performance limits of integrated circuit simulation on a dedicated minicomputer system," ERL Memo, UCB/ERL M81/29, May 22, 1981.

[21] A. Vladimirescu and D.O.Pederson, "Circuit simulation on vector processors," ICCC 82 Proceedings, pp. 172-175, September 28 - October 1, 1982.

[22] M.M.Denneau, "The Yorktown simulation engine," Proceeding of the 19th Design Automation Conference, pp. 55-59, June 1982.

[23] K.Hwang, Computer Arithmetic: Principles, Architecture and Design. New York: John Wiley, 1979, chaps. 6 and 8.

END

FILMED

1-86

DTIC